In [ ]:
```
# Initialize Otter
import otter
grader = otter.Notebook("CT-07-OOP-Intro.ipynb")
```

Aufgabe 1 (Klassen und Objekte). Erstellen Sie eine *Klasse* Ihrer Wahl. Diese Klasse sollte mindestens zwei *Attribute* und zwei *Methoden* beinhalten. Erzeugen Sie mehrere *Objekte* Ihrer *Klasse*.

In [1]:
```
# BEGIN SOLUTION
class Student():
    def __init__(self, name, age):
        self.name = name
        self.age = age

    def say_hello(self):
        print(f'Hello my name is {self.name} and I am {self.age} years old.')

student1 = Student('Daniel', 23)
student2 = Student('Maria', 24)
student1.say_hello()
student2.say_hello()
# END SOLUTION
```

```
Hello my name is Daniel and I am 23 years old.
Hello my name is Maria and I am 24 years old.
```

Aufgabe 2 (Klassen und Objekte). Erklären Sie in Ihren Worten den Unterschied zwischen einer Klasse und einem Objekt.

Type your answer here, replacing this text.

Eine *Klasse* ist die Blaupause bzw. die Bauanleitung und Funktionsbeschreibung ihrer *Objekte*. Sie definiert einen *zusammengesetzten Datentyp* inklusiver seiner *Methoden* wohingegen *Objekte* Werte von jenem Datentyp darstellen.

Aufgabe 3 (Klassen und Objekte). Folgender Code modelliert Rechtecke und Punkte und implementiert eine Funktion `contains` welche prüft ob sich ein Punkt innerhalb eines Rechtecks befindet. Wandeln Sie den Code in einen *objekt-orientierten* Code um.

1. Erstellen Sie eine Klasse `Rectangle` mit *Attributen* `x`, `y`, `width` und `height` und einer Funktion `contains`
2. Erstellen Sie eine Klasse `Point` mit *Attributen* `x`, `y`
3. Realisieren Sie geeignete Initialisierungen, d.h. `__init__`.
4. Realisieren Sie geeignete `__str__`-Methoden

In [2]:
```
def contains(rect, point):
    if point[0] < rect['x'] or point[1] < rect['y']:
        return False
    if point[0] > rect['x'] + rect['width'] or point[1] > rect['y'] + rect['height']:
        return False
    return True

point1 = (31, 41)
point2 = (-3, 50)
rect = dict(x=10, y=12, width=100, height=30)
print(contains(rect, point1))
print(contains(rect, point2))
```

```
True
False
```

In [3]:
```
class Rectangle():
    # BEGIN SOLUTION
    def __init__(self, x, y, width, height):
        self.x = x
        self.y = y
        self.width = width
        self.height = height

    def __str__(self):
        return f'x: {self.x}, y: {self.y}, width: {self.width}, height: {self.height}'

    def contains(self, point):
        if point.x < self.x or point.y < self.y:
            return False
        if point.x > self.x + self.width or point.y > self.y + self.height:
            return False
        return True
    # END SOLUTION

class Point():
    # BEGIN SOLUTION
    def __init__(self, x, y):
        self.x = x
        self.y = y

    def __str__(self):
        return f'({self.x}, {self.y})'
    # END SOLUTION

rect = Rectangle(x=10, y=12, width=100, height=30)
point1 = Point(31, 41)
point2 = Point(-3, 50)
print(rect.contains(point1))
print(rect.contains(point2))
```

```
True
False
```

```
In [4]: print(Rectangle(x=10, y=12, width=100, height=30))
        print(Point(31, 41))
```

```
x: 10, y: 12, width: 100, height: 30
(31, 41)
```

Aufgabe 4 (Vererbung und Komposition). Unser Rechteck ist durch einen Startpunkt `x`, `y` und durch die Breite `width` und Höhe `height` eindeutig definiert. Ein Punkt widerum durch `x` und `y`. Würden Sie zustimmen, dass deshalb ein Punkt eine Abstraktion eines Rechtecks ist?

Type your answer here, replacing this text.

Nein! Ein Punkt ist keine Abstraktion eines Rechtecks. Stattdessen ist ein Rechteck eine Komposition (Zusammensetzung) aus einem Punkt und zwei Längen (Breite und Höhe).

Aufgabe 5 (Klassen und Objekte). Ändern Sie Ihre Klasse `Rectangle` ab.

1. Nutzten Sie anstatt `x`, `y` einen `point` des Typs `Point`
2. Führen Sie Methoden `get_x1`, `get_y1`, `get_x2`, `get_y2` ein, welche Ihnen `point.x`, `point.y` sowie `point.x + width` und `point.y + height` zurückliefern.
3. Machen Sie alle *Attribute* privat.

Vermeiden Sie dabei doppelten Code!

```
In [5]: class Rectangle():
            # BEGIN SOLUTION
            def __init__(self, point, width, height):
                self.__point = point
                self.__width = width
                self.__height = height

            def get_x1(self):
                return self.__point.x

            def get_y1(self):
                return self.__point.y

            def get_x2(self):
```

```
            return self.get_x1() + self.__width

        def get_y2(self):
            return self.get_y1() + self.__width

        def __str__(self):
            return f'{self.__point}, width: {self.__width}, height: {self.__height}'

        def contains(self, point):
            if point.x < self.get_x1() or point.y < self.get_y1():
                return False
            if point.x > self.get_x2() or point.y > self.get_y2():
                return False
            return True
        # END SOLUTION
```

```
In [6]: rPoint = Point(x=10, y=12)
        rect = Rectangle(rPoint, width=100, height=30)
        point1 = Point(31, 41)
        point2 = Point(-3, 50)
        print(rect.contains(point1))
        print(rect.contains(point2))
        print(rect)
        print(point1)
```

```
True
False
(10, 12), width: 100, height: 30
(31, 41)
```

Aufgabe 6 (Klassen und Objekte). Erweitern Sie Ihre Klasse `Point` um die Methode `add(point)` und `sub(point)` sodass

```
Point(1,1).add(Point(2,3))
```

einen **neuen** `Point(3,4)` zurückliefert. Implementieren Sie `sub` (die Subtraktion) entsprechend.

```
In [7]: class Point():
            def __init__(self, x, y):
                self.x = x
                self.y = y

            def __str__(self):
                 return f'({self.x}, {self.y})'
```

```
        # BEGIN SOLUTION
        def add(self, point):
            return Point(self.x + point.x, self.y + point.y)

        def sub(self, point):
            return Point(self.x - point.x, self.y - point.y)
        # END SOLUTION
```

```
In [8]: p1 = Point(1,1)
        p2 = Point(2,3)
        p3 = p1.add(p2)
        p4 = p1.sub(p2)
        print(f' {p1}.add({p2}) == {p3}')
        print(f' {p1}.sub({p2}) == {p4}')
```

```
 (1, 1).add((2, 3)) == (3, 4)
 (1, 1).sub((2, 3)) == (-1, -2)
```

Aufgabe 7 (Polymorphie). Finden Sie heraus (Google ist Ihr Freund) wie Sie den `+` und `-`-Operator **überladen** (engl. overload) können, sodass Sie anstatt

```
Point(1,1).add(Point(2,3))
```

das folgende schreiben können

```
Point(1,1) + Point(2,3)
```

bzw. anstatt

```
Point(1,1).sub(Point(2,3))
```

das folgende schreiben können

```
Point(1,1) - Point(2,3)
```

Passen Sie Ihre Klasse `Point` entsprechend an!

```
In [9]: class Point():
            def __init__(self, x, y):
                self.x = x
                self.y = y

            def __str__(self):
                return f'({self.x}, {self.y})'

            # BEGIN SOLUTION
            def __add__(self, point):
                return Point(self.x + point.x, self.y + point.y)

            def __sub__(self, point):
                return Point(self.x - point.x, self.y - point.y)
            # END SOLUTION
```

```
In [10]: p1 = Point(1,1)
         p2 = Point(2,3)
         p3 = p1 + p2
         p4 = p1 - p2
         print(f' {p1} + {p2} == {p3}')
         print(f' {p1} - {p2} == {p4}')
```

```
 (1, 1) + (2, 3) == (3, 4)
 (1, 1) - (2, 3) == (-1, -2)
```

Aufgabe 8 (Veränderlichkeit). Sind Ihre Objekte von der Klasse `Point` veränderlich oder unveränderlich? Sofern Sie veränderlich sind, wie könnten Sie sicherstellen, dass sie unveränderlich sind? Führen Sie diese Änderungen an der Klasse `Point` durch.

Type your answer here, replacing this text.

Sofern unsere Attribute `x` und `y` nicht *privat* sind, sind unsere Objekte veränderlich. Denn

```
In [11]: p1 = Point(1,1)
         p1.x = 4
         print(p1)
```

```
(4, 1)
```

ist erlaubt. Wir müssen unsere Attribute demnach *privat* machen. Da keine Methode irgendein Attribut des Objekts verändert, wäre es dann unveränderlich.

```
In [12]: class Point():
             # BEGIN SOLUTION
             def __init__(self, x, y):
                 self.__x = x
                 self.__y = y

             def __str__(self):
                 return f'({self.__x}, {self.__y})'

             def __add__(self, point):
                 return Point(self.__x + point.__x, self.__y + point.__y)

             def __sub__(self, point):
                 return Point(self.__x - point.__x, self.__y - point.__y)
             # END SOLUTION
```

```
In [13]: p1 = Point(1,1)
         p1.x = 4
         print(p1)
         print(p1 + Point(2,2))
```

```
(1, 1)
(3, 3)
```

To double-check your work, the cell below will rerun all of the autograder tests.

```
In [ ]: grader.check_all()
```

0.1 Submission

Make sure you have run all cells in your notebook in order before running the cell below, so that all images/graphs appear in the output. The cell below will generate a zip file for you to submit.

```
In [ ]: grader.export(pdf=False, force_save=True)
```