Übung: Objektorientierte Programmierung

Aufgabe 1: Modelliere einen Würfel

Erstelle eine Klasse *Cube*, mit der du einen Würfel modellierst! Die Würfel-Klasse soll als Eigenschaft die Länge einer Würfel-Seite besitzen. Darüber hinaus soll die Klasse auch zwei Methoden haben: die Methode `volume()` berechnet das Volumen und gibt es aus, die Methode `surface()` berechnet die Oberfläche und gibt sie aus.

In [16]:
```
class Cube():

    def __init__(self, side):
        self.side = side

    def surface(self):
        print(self.side ** 2 * 6)

    def volume(self):
        print(self.side ** 3)


# danach erzeugen wir eine Instanz deiner Cube-Klasse
a = Cube(3)

# und testen die Methoden
a.surface()
a.volume()
```

```
54
27
```

Erwartete Ausgabe:

```
54
27
```

Aufgabe 2: Modelliere ein Kugel

Die Kugel-Klasse soll als Eigenschaft den Radius übergeben bekommen. Zudem soll sie - ähnlich wie der Würfel - zwei Methoden haben: `surface()` um den Oberflächeninhalt zu berechnen, `volume()` um das Volumen zu berechnen.

Damit du diese Berechnungen durchführen kannst, benötigst du die Kreiszahl Pi. Diese steht dir nach einem `import math` unter `math.pi` zur Verfügung (was der `import` - Befehl genau macht, schauen wir uns noch später im Kurs an).

Die Formeln für den Oberflächeninhalt / das Volumen einer Kugel darfst du im Internet nachgucken.

In [6]:
```
import math
print(math.pi)
```

```
3.141592653589793
```

In [8]:
```
import math

class Ball():

    def __init__(self, radius):
        self.radius = radius

    def surface(self):
        print(4 * math.pi * self.radius ** 2)

    def volume(self):
        print(4 / 3 * math.pi * self.radius ** 3)


b = Ball(4)

# und testen die Methoden
b.surface()
b.volume()
```

```
201.06192982974676
268.082573106329
```

Erwartete Ausgabe (es reicht, wenn die Zahlenwerte auf ein paar Nachkommastellen genau sind):

```
201.06192982974676
268.082573106329
```

Aufgabe 3: Modelliere ein Konto

a.)

Erstelle die Konto-Klasse *Account* mit der Eigenschaft Kontostand *credits*! Diese Eigenschaft wird mit einem Startkapital initialisiert. Die Methode `display()` soll den aktuellen Kontostand ausgeben.

In [8]:
```
class Account():

    def __init__(self, start):
        self.credits = start

    def display(self):
        print(self.credits)

Kunde111 = Account(500)
Kunde111.display()
```

```
500
```

Erwartete Ausgabe:

```
500
```

b.)

Ergänze die Klasse *Account* um zwei Methoden (`pay_in()` zum Einzahlen, `withdraw()` zum Abheben), so dass du Geldbeträge einzahlen und abbuchen kannst und der Kontostand

entsprechend angepasst wird!

Du sollst nur Geld abheben können, solange auch Geld auf dem Konto ist. Ein Dispo-Kredit wird nicht gewährt. In dem Fall soll eine Fehlermeldung ausgegeben werden, in der steht, wieviel Geld maximal abgebucht werden kann.

In [17]:
```
class Account():

    # Ergänze hier deinen Code. Du darfst den Code aus a)
    # natürlich hierhin übernehmen.

    def __init__(self, start):
        self.credits = start

    def display(self):
        print(self.credits)

    def pay_in(self, money):
        self.credits += money

    def withdraw(self, money):
        if self.credits >= money:
            self.credits -= money
        else:
            print("Du kannst nur noch " + str(self.credits) + "€ abheben!")

Kunde111 = Account(500)
Kunde111.display()
Kunde111.pay_in(40)
Kunde111.display()
Kunde111.withdraw(25)
Kunde111.display()
Kunde111.withdraw(600)
```

```
500
540
515
Du kannst nur noch 515€ abheben!
```

Erwartete Ausgabe:

```
500
540
515
Du kannst nur noch 515€ abheben!
```

c.)

Bislang ist das Konto noch ungeschützt - wir brauchen eine PIN! Ergänze in der Klasse die Eigenschaft *pin*! So wie mit dem Startkapital soll das Konto auch mit einer PIN initialisiert werden.

Von nun an muss man beim Geldabheben nicht nur den Betrag, sondern auch die PIN angeben: Nur wenn die PIN mit der des Kontos übereinstimmt, kann auch Geld abgebucht werden, ansonsten kommt es zu einer Fehlermeldung!

In [11]:

```
class Account():
    # Ergänze hier deinen Code. Du darfst den Code aus b)
    # natürlich hierhin übernehmen.

    def __init__(self, start, pin):
        self.credits = start
        self.pin = pin

    def display(self):
        print(self.credits)

    def pay_in(self, money):
        self.credits += money

    def withdraw(self, money, pin):
        if pin == self.pin:
            if self.credits > money:
                self.credits -= money
            else:
                print("Du kannst nur " + str(self.credits) + "€ abheben!")
        else:
            print("Falsche PIN! Konto gesperrt! Du bist verhaftet! Hände hoch!")

Kunde111 = Account(500, "1234")
Kunde111.display()
Kunde111.pay_in(40)
Kunde111.display()
Kunde111.withdraw(25, "1234")
Kunde111.display()
Kunde111.withdraw(600, "12345")
```

```
500
540
515
Falsche PIN! Konto gesperrt! Du bist verhaftet! Hände hoch!
```

Erwartete Ausgabe:

```
500
540
515
Falsche PIN! Konto gesperrt! Du bist verhaftet! Hände hoch!
```

Aufgabe 4: Modelliere einen Zug

a.)

Jetzt wirst du Zugobjekte bauen! Erstelle die Klasse *Train*, die mit den Eigenschaften *route* und *position* initialisiert wird! Bei *route* handelt es sich um eine Liste mit den Haltebahnhöfen des Zuges. *position* steht für den Index des Bahnhofs aus der Liste, an dem sich der Zug gerade befindet bzw. von dem er zuletzt abgefahren ist (wo genau sich der Zug auf der Strecke zwischen zwei Bahnhöfen befindet, interessiert uns hier nicht). Mit der Methode `show_station()` soll der Name dieses Bahnhofs ausgegeben werden.

In [12]:
```
class Train():

    def __init__(self, route, start):
        self.route = route
        self.position = start

    def show_station(self):
        print(self.route[self.position])

orientexpress = Train(["Paris", "Budapest", "Bukarest", "Istanbul"], 0)
orientexpress.show_station()
```

```
Paris
```

Erwartete Ausgabe:

```
Paris
```

b.)

Bisher sitzt ein Zug der Klasse *Train* noch in seinem Startbahnhof fest. Ergänze nun zwei Methoden `move()` und `move_back()`, mit denen man einen Zug auf seiner Route zur nächsten bzw. zur vorherigen Station bewegen kann, sofern es diese Station auf der Route gibt. Der Zug darf seine Route nicht verlassen!

In [14]:
```
class Train():

    # Ergänze hier deinen Code. Du darfst natürlich deinen Code aus
    # Teilaufgabe a) hierhin übernehmen.

    def __init__(self, route, start):
        self.route = route
        self.position = start

    def show_station(self):
        print(self.route[self.position])

    def move(self):
        if self.position < len(self.route) - 1:
            self.position += 1
        else:
            print("Endstation! Alle aussteigen!")

    def move_back(self):
         if self.position > 0:
            self.position -= 1

orientexpress = Train(["Paris", "Budapest", "Bukarest", "Istanbul"], 0)
orientexpress.show_station()
orientexpress.move()
orientexpress.show_station()
orientexpress.move()
orientexpress.show_station()
orientexpress.move()
orientexpress.show_station()
orientexpress.move()
orientexpress.move_back()
orientexpress.show_station()
```

```
Paris
Budapest
Bukarest
Istanbul
Endstation! Alle aussteigen!
Bukarest
```

Erwartete Ausgabe:

```
Paris
Budapest
Bukarest
Istanbul
Endstation! Alle aussteigen!
Bukarest
```

c.)

Die Route soll nachträglich bearbeitet werden können, indem man mit einer Methode `bypass_station()` einen anzugebenden Haltebahnhof von der Route entfernt. Der Zug soll dann sicherheitshalber an den Start der Route versetzt werden, sofern er sich nicht schon dort befindet.

Tipp: Erinnere dich an die Methoden für Listen! :-)

In [15]:

```
class Train():

    # Ergänze hier deinen Code. Du darfst natürlich deinen Code aus
    # Teilaufgabe b) hierhin übernehmen.

    def __init__(self, route, start):
        self.route = route
        self.position = start

    def show_station(self):
        print(self.route[self.position])

    def move(self):
        if self.position < len(self.route) - 1:
            self.position += 1
        else:
            print("Endstation! Alle aussteigen!")

    def move_back(self):
         if self.position > 0:
            self.position -= 1

    def bypass_station(self, station):
        if station in self.route:
            self.route.remove(station)
            self.position = 0

orientexpress = Train(["Paris", "Budapest", "Bukarest", "Istanbul"], 0)
orientexpress.bypass_station("Budapest")
orientexpress.move()
orientexpress.show_station()
```

```
Bukarest
```

Erwartete Ausgabe:

```
Bukarest
```

Gut gemacht :-)