Home

Tartalomjegyzék

Exercises

Complex

You have to add methods to the Complex class from the lecture:

```
class Complex(object):
    def __init__(self, real, imaginary):
        self.re = real
        self.im = imaginary

    def __add__(self, z2):
        new_re = self.re + z2.re
        new_im = self.im + z2.im
        return Complex(new_re, new_im)

    # used by print
    def __str__(self):
        return str(self.re) + " + " + str(self.im) + "i"

z1 = Complex(4, 3)
z2 = Complex(-2, 1)
z3 = z1 + z2

print(z3)
```

- Implement the subtraction, multiplication and division methods (__sub__, __mul__, __truediv__).
- Also implement the `norm` method which returns the length of the complex number.
- Improve the __str__ method to handle numbers like this for example:

```
2 - 4i
```

```
5i
2
```

For testing use this (or you can use other tests):

```
k1 = Complex(4, 3)
k2 = Complex(-2, 1)
k3 = Complex(4, 1)

print(k1 + k2)
print(k1 - k3)
print(k2 * k1)
print(k3 / k1)
print(k1.norm())
```

Reservation 1

In this exercise you are working on a seat reservation program for a movie theater. You will handle the seat reservations of the customers.

Create a **Reservation** class which stores the reservation of a customer.

In this exercise create the class with a constructor (__init__). The class will have two members which are also the parameters of the constructor:

- **name** the name of the customer
- **seats** a list of seats what the customer reserved. A seat is marked whit a unique string, like "http://wiki.math.bme.huE12"http://wiki.math.bme.hu.

The only thing that you should write is a constructor. Store the given two values in self.

Hint: Like any method, the first parameter should be self.

Reservation 2

In this exercise add a method to the previous **Reservation** class. The method should be called **not_taken** and should have one parameter (except self): **choices** a list of seats, the possible seats to take.

The method should return a list of seats that are not taken from **choices**. The order of the seats should be the same as the input, but a subset of them.

Hint: Use the **self.seats** to determine the already taken seats and you should exclude those from **choices**. Use the previous exercise as a starting point.

Reservation 3

In this exercise we will use the previous **Reservation** class. Write a function called **reservation_3**, the parameters:

- **reservations**, a list of objects of type Reservation, the list of active reservations
- **all_seats**, a list of all seats in the theater

The function should return a list of free seats. A seat is free if it is not taken by any of the reservations.

Hint: Use the **Reservation** class and its **not_taken** method. Copy the **Reservation** class from the previous exercise.

Reservation 4

You should write a method that modifies the reserved seats. Write a new method in the previous **Reservation** class called **change**, its only parameter (except self) should be:

- **new_seats**, a list of seats, this should be the value of the member seats after the call of the method.

Since this method modifies the object, don't return anything, just modify the member **self.seats**.

Reservation 5

We will use the previous **Reservation** class to handle the reservation of a new customer. Write a function called **reservation_5** with 4 parameters:

- **reservations**, a list of previous reservations (each object of class **Reservation**).
- **all_seats**, a list of all seats in the theater
- **name**, name of a new customer who wants to buy tickets.
- **count**, the number of seats he/she wants to reserve.

The function should search for count many consecutive, empty seats. If you find the appropriate seats, then return an object of type Reservation which contains the new reservation (name and the reserved seats). If you cannot find such seats then return a None.

Hint: Use the solutions of the previous exercises.

Whowins

In the code below you can see a **Match** class which stores data of matches of some game. In a match there are two teams: A and B and both get some points.

There are two methods already written: a constructor and one for accounting the points.

One method is missing, you have to write a method called **whowins** which tells who wins the match.

The method should have one parameter (except self):

- **limit**, a number which determines a minimum point value. Under this limit nobody wins

The method should return te name of the winning team or 'nobody' if there are no winners.

The winning team is the one that reached limit points. If both teams reached that point, then team A wins (even if B has more points). If none of them reached that limit, then nobody wins.

Code:

```
class Match:
    def __init__(self, A_name, B_name):
        self.A_name = A_name
        self.B_name = B_name
        self.A_point = 0
```

```
        self.B_point = 0

    def point(self, name, number):
        if   self.A_name == name:
            self.A_point += number
        elif self.B_name == name:
            self.B_point += number
        else:
            raise ValueError("http://wiki.math.bme.huUnknown team!"http://wiki.math.bme.hu)
```