Tartalomjegyzék

Problem 1 : range

Write an iterable class like **range**, but without returning a whole list, but storing only the actual element.

```
class Range:
     def __init__( ... ):
         ...
     def __iter__( ... ):
         ...
     def __next__( ... ):
         ...
```

- Its constructor should have one parameter: a number or a string. The iteration should go up that number, from 0 with 1 steps.
- If the number is not positive, then the iteration should take 0 steps.
- If you get a string then try to convert it to a number. It it cannot be converted, then raise a **ValueError** exception.
    - ♦ If it is a valid integer, then calculate with that.
- If you get the string **"http://wiki.math.bme.huinf"http://wiki.math.bme.hu** then make the iteration go endless (infinite loop)!

Problem 2: Shapes

Write a class called **Shape**.

- Let it have two members: **x** and **y**, the coordinates of the shape on the plane (center of mass).
- Define a **move** method, with one parameter **v**: a list of length 2, a vector to translate the shape with. After this method the coordinates should be changed.

Define the following classes as children of **Shape**:

- **Ellipse** with additional parameters (except the *(x, y)* coordinates): **a** and **b** the *x* and *y* axes radii
- **Rectangle** with additional parameters (except the *(x, y)* coordinates) **a** and **b** the length of the sides

Write an **area** method for both, which calculates the area!

Define an **equation** method for printing the equation of the **Ellipse**! Something like:

```
((x-1)/2)^2 + ((y-2)/3)^2 = 1
```

Problem 3: moduloz_init

Define a class called Moduloz representing modulo n numbers (integers). For example in modulo 5:

```
4 + 3 = 2 (because 7 % 5 = 2)
2 - 3 = 4 (because -1 % 5 = 4)
4 * 3 = 2 (because 12 % 5 = 2)
```

You don't have to implement the operations yet, just define the __init__ and the __str__ methods.

In the constructor you will have two parameters, except self. The first one is the base of the modulo, the second one is the actual number.

The base will be a positive integer, the value will be an integer.

The __str__ should return a string, containing the value.

For example:

```
a = ModuloZ(5, 7)
print a
```

Should print:

```
2
```

Problem 4: moduloz_operations

Implement the __add__, __sub__, __mul__ methods for the previous Moduloz class!

For example in modulo 5:

```
4 + 3 = 2 (because 7 % 5 = 2)
2 - 3 = 4 (because -1 % 5 = 4)
4 * 3 = 2 (because 12 % 5 = 2)
```

Mind that the operations should return an object of class Moduloz, not an integer (int)! For example:

```
a = Moduloz(7, 9) b = Moduloz(7, 12)
print a + b print a - b print a * b
```

should print:

```
0 4 3
```

In the test outputs you can see the sum, difference and the product of the two input numbers.

Hint

Use the previous exercise as a starting point.

Problem 5: matrix_init=

Define a class called Matrix for representing matrices.

You have to implement the __init__ and __str__ methods. The constructor has one parameter (except self), a list of list of numbers. The elements of the matrix. The __str__ should return a multi-line string, containing the matrix in a tabular-like format. For example:

```
m = Matrix([[1, 2], [13, 4], [5, 6]])  print m
```

should print this:

```
 1   2
13   4
 5   6
```

The numbers are padded to the right in 4 characters width. There are 3 spaces before each element, except the 13 because there are 2 spaces there.

Problem 6: matrix_operations

Implement the __add__, __sub__, __mul__ methods for the previous Matrix class.

The matrices will be square shaped, so every operation is compatible.

For example:

```
m1 = Matrix([[1, 2], [3, 4]]) m2 = Matrix([[1, 0], [0, 2]])  print a + b
```

should print this:

```
   2   2
   3   6
```

In the test you can see the sum, difference and the dot product of the two input matrices. Hint

Use the previous exercise as a starting point.