previous up next

Tartalomjegyzék

hazi@math.bme.hu

- `reservation_6`
  - you have to do the previous 5 first
- `divisor_iterate`

Iterables

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)!

Polymorphism

Shapes on a canvas

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
```

Inheritance and constructors

If the child class (e.g. Ellipse) you want to call the parents class' constructor then you have two ways:

```
class B(A):
    def __init__(self, x, y, a, b):
         A.__init__(self, x, y)
         # OR
         super().__init__(x, y)
```

The first explicitly calls the parents `__init__`

```
A.__init__(self, x, y)
```

The second one calls the parent of **B** which happens to be **A**:

```
super().__init__(x, y)
```

Canvas

Define a class called **Canvas**

- Its only member variable should be a list of Shapes: **shapes**. This stores several Shape objects.
- Define an **add** method which adds a new shape to the canvas!
- Make this class **iterable**! Write the **__iter__(self)** and the **__next__(self)** methods, as seen on the lecture.
- Define a **crop** method in the following way.
    - ♦ Its two parameters should be two coordinates: a top-left corner and a bottom-right corner.
    - ♦ The function should return the list of Shapes which entirely fit in that rectangle.

    - For this it is best to implement a **box()** method on both **Ellipse** and **Rectangle** which returns a bounding box of the shape:

a list of two coordinates, the top-left corner and a bottom-right corner of the object.

Chess

Define a class called **Piece**. This will represent a chess piece (a figure on the chessboard). Store its position and its color (Black/White) and its **__str__** function should write the position like A2, G3 etc. Store the position as integers.

- Define child classes **King** and **Pawn**!
- Implement their **__str__** method which also prints the type of the figure. (You can implement it in the parent class, too)
    - Look the unicode characters of the figures: [1]
    - Or their one letter names
- Every child class should have a **.move(pos)** method, where **pos** is a string like A3, G2 etc. Move the piece to that position if it is accepted by the chess rules. If the move was successful return **True** otherwise **False**!
- Define a **PieceMoveError** exception. If the move is not allowed, then raise this exception!

Implement the **Board** class. Store the list of figures.

- Make an **add** method which puts a figure on the table.
- Implement a **move(pos1, pos2)** method in **Board** which moves a figure from one position to the other, if the move is allowed (handle the **PieceMoveError** exception).
    - Mind that there may be an other figure on the **pos2** position. If the target position is occupied with the same player's figure, then the move is not allowed. If the target is the other player's figure, then remove their figure and replace it with the new figure (its a *capture*).
    - Its easier if you implement an **.occupied(pos)** method first, which tells whether a given position is occupied or not.
- Implement the other pieces: **Knight, Rook, Bishop** and **Queen**!
    - Start with the **Rook**, its move is the simplest. Mind that the figure cannot pass through existing figures!
    - Implement the **Knight** class! The Knight and Queen can jump through other figures, so its not a problem for them.
    - After the bishop its not hard to implement **Queen**.
- Define a **check** method in Board. Return the color of the player who's King is in a chess position, or empty string of there is no chess position!
    - Modify the Board's move method not to allow self-checks (when the active player makes its own King into a chess)!
- Implement the **__str__** method of **Board**!

After all these, implement the **start** method which assigns the board to a starting position.

CLI

- Make a chess game in command line: read the moves of the players with `input` and print the board after each step. The white player starts and mind that the same player cannot move twice.
    - It's better to use algebraic notation (Bf5, Qc3, Ne2, Kcd4, Kxd5 etc.)
- Try it from command line