Universidad Nacional San Agustin de Arequipa

Facultad de Ingenieria de Produccion y Servicios

Escuela Profesional de Ingenieria de Sistemas

Curso:

Programacion Web 2

Profesor:

Carlo Jose Luis Corrales Delgado

Tema:

Laboratorio 5 - Python

Integrantes:

Cuno Cahuari Armando Steven

Repositorio:

https://github.com/SteveArms/pythonPWEB.git

2024

Informe Nro 5 - Python

La actividad consiste en utilizar un objeto llamado Picture, que proporciona diversos métodos de clase necesarios para crear diferentes formas de las imágenes solicitadas. Primero, es necesario implementar estos métodos en la clase Picture. Luego, se deben desarrollar los códigos necesarios para utilizar adecuadamente estos métodos en las imágenes solicitadas.

Metodos Picture - Python

Metodo verticalMirror

El método 'verticalMirror' creará un reflejo vertical de una imagen, similar a un espejo. Para lograr esto, utilizaremos una nueva lista vacía donde copiaremos los datos del original al nuevo mediante una iteración que recorre las cadenas que conforman la figura. Cada fila se agregará en sentido inverso utilizando 'value[::-1]'. Una vez finalizada la iteración, retornaremos esta nueva lista convertida en un objeto 'Picture', permitiendo así el uso de los métodos de clase proporcionados.

```
14    def verticalMirror(self):
15      """ Devuelve el espejo vertical de la imagen """
16      vertical = []
17      for value in self.img:
18          vertical.append(value[::-1])
19      return Picture(vertical)
```

Figure 1: Metodo verticalMirror

Metodo horizontalMirror

El método 'horizontalMirror' creará un reflejo horizontal de la imagen, simulando un espejo a lo largo de una línea horizontal. Utilizaremos una nueva lista que contendrá las cadenas invertidas. Para ello, recorreremos las filas de la imagen de manera descendente utilizando 'range', accediendo a los últimos elementos primero y agregándolos a la nueva lista en orden inverso. Una vez completada la iteración, devolveremos esta nueva lista como un objeto 'Picture', permitiendo así el uso de los métodos de clase proporcionados.

```
21    def horizontalMirror(self):
22      """ Devuelve el espejo horizontal de la imagen """
23      nuevo = []
24      for i in range(len(self.img) - 1 , -1, -1):
25        nuevo.append(self.img[i])
26      return Picture(nuevo)
```

Figure 2: Metodo horizontalMirror

Metodo negative

El método 'negative' cambiará los colores de la figura a sus opuestos. Para ello, utilizaremos el método 'invColor', que ya está implementado y recibe caracteres, devolviendo su color opuesto utilizando un diccionario.
El método 'negative' creará una nueva lista con los cambios aplicados mediante una doble iteración: una para recorrer las filas y otra para recorrer los caracteres dentro de cada fila. Durante la segunda

iteración, los caracteres de la cadena se pasarán por el método ‘invColor‘, y una nueva variable ‘cadenaFinal‘ irá acumulando estos caracteres invertidos. Al finalizar la segunda iteración, ‘cadenaFinal‘ se agregará a la lista. Finalmente, se devolverá esta lista nueva convertida en un objeto ‘Picture‘.

```
28    def negative(self):
29      nuevo = []
30      for i in range(len(self.img)):
31        cadenaFinal = ""
32        cadena = self.img[i]
33        for x in range(len(cadena)):
34          cadenaFinal += self._invColor(cadena[x])
35        nuevo.append(cadenaFinal)
36      return Picture(nuevo)
```

Figure 3: Metodo negative

```
 9    def _invColor(self, color):
10      if color not in inverter:
11        return color
12      return inverter[color]
```

Figure 4: Metodo auxiliar invColor

Metodo join

El método ‘join‘ colocará un objeto a la derecha del objeto actual. Para lograr esto, utilizaremos una lista que se construirá mediante una iteración según el tamaño de los objetos. Durante la iteración, combinaremos los valores de las filas del objeto actual con las filas del objeto que se va a añadir. Una vez realizada esta combinación, se agregará a la lista. Finalmente, se retornará esta nueva lista convertida en ‘Picture‘.

```
38    def join(self, p):
39      nuevo = []
40      for i in range(len(self.img)):
41        nuevo.append(self.img[i] + p.img[i])
42      return Picture(nuevo)
```

Figure 5: Metodo join

Metodo up

El método 'join' colocará un objeto encima del objeto actual. Para lograr esto, primero realizaremos una iteración sobre las filas del objeto que se va a colocar encima, agregándolas a una nueva lista. Luego, en una segunda iteración, agregaremos las filas del objeto actual a esta nueva lista. Finalmente, retornaremos esta lista convertida en un objeto 'Picture'.

```
44    def up(self, p):
45      nuevo = []
46      for i in range(len(p.img)):
47        nuevo.append(p.img[i])
48      for x in range(len(self.img)):
49        nuevo.append(self.img[x])
50      return Picture(nuevo)
```

Figure 6: Metodo up

Metodo under

El método 'under' colocará un objeto sobre el objeto actual. Para lograr esto, utilizaremos una doble iteración: una que recorra las filas del objeto que se va a colocar encima y otra que recorra los caracteres de cada fila.
Durante la iteración, crearemos una nueva variable 'cadenaFinal' para almacenar los nuevos datos. Si el carácter en la posición del objeto a colocar es vacío, agregaremos el carácter correspondiente del objeto actual; de lo contrario, agregaremos el carácter del objeto a colocar encima.
Al finalizar la segunda iteración, 'cadenaFinal' se agregará a una nueva lista. Finalmente, retornaremos esta lista convertida en un objeto 'Picture'.

```
52    def under(self, p):
53      nuevo = []
54      for i in range(len(p.img)):
55        cadenaFinal = ""
56        cadena = p.img[i]
57        for x in range(len(cadena)):
58          if(cadena[x] == ' '):
59            cadenaAuxiliar = self.img[i]
60            cadenaFinal += cadenaAuxiliar[x]
61          else :
62            cadenaFinal += cadena[x]
63        nuevo.append(cadenaFinal)
64      return Picture(nuevo)
```

Figure 7: Metodo under

Metodo horizontalRepeat

El método 'horizontalRepeat' repetirá un objeto hacia la derecha un número específico de veces, el cual se pasará como argumento. Para lograr esto, utilizaremos una doble iteración: la primera recorrerá las filas y la segunda se encargará de repetir cada fila el número de veces especificado.
Durante la segunda iteración, utilizaremos una lista vacía y una variable auxiliar para agregar repetidamente el contenido de la fila. Una vez que se haya repetido el contenido de la fila el número de veces deseado, lo agregaremos a la lista.
Finalmente, retornaremos la lista convertida en un objeto 'Picture'.

```
66    def horizontalRepeat(self, n):
67      """ Devuelve una nueva figura repitiendo la figura actual al costado
68          la cantidad de veces que indique el valor de n """
69      nuevo = []
70      for i in range(len(self.img)):
71        cadena = ""
72        for y in range(n):
73          cadena += self.img[i]
74        nuevo.append(cadena)
75      return Picture(nuevo)
```

Figure 8: Metodo horizontalRepeat

Metodo verticalRepeat

El método 'verticalRepeat' repetirá un objeto hacia arriba un número específico de veces, el cual se pasará como argumento. Utilizaremos una doble iteración: la primera iteración repetirá la acción el número de veces especificado y la segunda iteración recorrerá las filas del objeto.
Durante la segunda iteración, agregaremos las filas correspondientes del objeto a una nueva lista. Una vez completada la primera iteración, retornaremos esta lista convertida en un objeto 'Picture'.

```
77    def verticalRepeat(self, n):
78      nuevo = []
79      for i in range(n):
80        for y in range(len(self.img)):
81          nuevo.append(self.img[y])
82      return Picture(nuevo)
```

Figure 9: Metodo verticalRepeat

Metodo rotate

El método 'rotate' girará el objeto 90 grados en sentido antihorario. Para lograr esto, utilizaremos una lista vacía como referencia y una doble iteración. La primera iteración recorrerá los caracteres de cada cadena correspondiente a una fila del objeto, de los últimos caracteres hacia los primeros. La segunda iteración recorrerá las filas del objeto.
Utilizaremos una variable 'cadenaNueva' que agregará los caracteres correspondientes al nuevo orden de las filas, ajustando la orientación. Al finalizar ambas iteraciones, agregaremos 'cadenaNueva' a la lista. Finalmente, retornaremos esta lista convertida en un objeto 'Picture'.

```
84    #Extra: Sólo para realmente viciosos
85    def rotate(self):
86      """Devuelve una figura rotada en 90 grados, puede ser en sentido horario
87      o antihorario"""
88      nuevo = []
89      for i in range(len(self.img[0]) - 1, -1 ,-1):
90        cadenaNueva = ""
91        for x in range(len(self.img)):
92          cadenaNueva += self.img[x][i]
93        nuevo.append(cadenaNueva)
94      return Picture(nuevo)
```

Figure 10: Metodo rotate

Dibujo de las figuras - Draw

1er Ejercicio

Para el primer ejercicio, primero importamos la biblioteca picture y sus respectivas dependencias para hacer uso de sus funcionalidades. Se nos pide crear un tablero de ajedrez de 2 x 2 con caballos en las diagonales de color opuesto.
Primero, creamos la fila inicial utilizando la pieza knight y la función join para colocar dos caballos uno al lado del otro. Aplicamos la función negative a uno de los caballos para obtener el color opuesto. Guardamos esta combinación en una variable llamada fila.
Luego, creamos la segunda fila aplicando la función negative en la variable fila para invertir los colores y obtener la segunda fila. Esta se almacena en una variable llamada fila2. A continuación, unimos las filas usando la función up para colocar fila2 encima de fila, formando así el tablero de 2 x 2.
Finalmente, utilizamos la función draw para visualizar el tablero creado.

```
1   #Importar
2   from chessPictures import *
3   from interpreter import draw
4   fila = knight.join(knight.negative())
5   fila2 = fila.negative()
6   draw(fila2.up(fila))
```

Figure 11: Codigo - Ejercicio 1

Figure 12: Draw - Ejercicio 1

2do Ejercicio

Para el segundo ejercicio, realizaremos un procedimiento similar al anterior, pero con la diferencia de que la fila inferior debe estar mirando del otro lado.
Primero, creamos la fila inicial utilizando la pieza knight y la función join para colocar dos caballos uno al lado del otro. Aplicamos la función negative a uno de los caballos para obtener el color opuesto. Guardamos esta combinación en una variable llamada fila.
Luego, creamos la segunda fila utilizando la función verticalMirror en la variable fila, lo que resulta en un espejo vertical de esta.
Para unir las filas, usamos la función up para colocar la segunda fila debajo de la primera, formando así el tablero.
Finalmente, utilizamos la función draw para visualizar el tablero creado.

```
1  #Importar
2  from chessPictures import *
3  from interpreter import draw
4  arriba = knight.join(knight.negative())
5  abajo = arriba.verticalMirror()
6  draw(abajo.up(arriba))
```

Figure 13: Codigo - Ejercicio 2

Figure 14: Draw - Ejercicio 2

3er Ejercicio

En el tercer ejercicio, debemos colocar cuatro reinas en una fila. Para lograr esto, utilizaremos el método horizontalRepeat, que permite repetir un objeto un número específico de veces hacia la derecha. En este caso, el objeto será una reina, y el método repetirá la reina cuatro veces. Luego, usaremos la función draw para visualizar el resultado.

```
1  #Importar
2  from chessPictures import *
3  from interpreter import draw
4
5  draw(queen.horizontalRepeat(4))
```

Figure 15: Codigo - Ejercicio 3

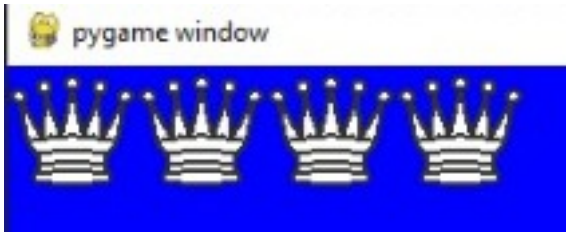

Figure 16: Draw - Ejercicio 3

4to Ejercicio

En el cuarto ejercicio, debemos mostrar una fila de 8 cuadrados siguiendo el esquema de un tablero de ajedrez. Para lograr esto, primero creamos una variable casillero que representa dos cuadrados, utilizando el método join para colocar un cuadrado al lado del otro, aplicando el método negative a uno de ellos para cambiar su color.
Luego, usaremos el método horizontalRepeat con el argumento 4 para repetir este par de cuadrados, obteniendo así una fila de 8 casilleros con el patrón de un tablero de ajedrez. Finalmente, utilizamos la función draw para visualizar el resultado.

```
1  #Importar
2  from chessPictures import *
3  from interpreter import draw
4
5  casillero = square.join(square.negative())
6  draw(casillero.horizontalRepeat(4))
```

Figure 17: Codigo - Ejercicio 4

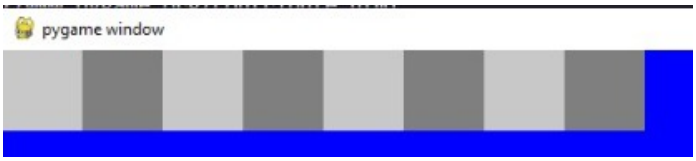

Figure 18: Draw - Ejercicio 4

5to Ejercicio

En el quinto ejercicio, se nos pide crear una fila de 8 cuadrados con el mismo esquema de un tablero de ajedrez, pero con los colores invertidos. Para lograr esto, reutilizaremos el código del ejercicio anterior y aplicaremos el método negative al resultado final de horizontalRepeat antes de visualizarlo con draw.

```
1  #Importar
2  from chessPictures import *
3  from interpreter import draw
4
5  casillero = square.join(square.negative())
6  draw(casillero.horizontalRepeat(4).negative())
```

Figure 19: Codigo - Ejercicio 5

Figure 20: Draw - Ejercicio 5

6to Ejercicio

En el sexto ejercicio, debemos crear un tablero de ajedrez de 8 x 4 en el cual la fila más baja debe ser la misma que en el ejercicio 5, respetando el patrón del ajedrez.
Primero, creamos una variable casilleroX2 que representa dos cuadrados alternados en color. Utilizamos los objetos square y el método join, aplicando negative a uno de ellos para cambiar su color. Luego, repetimos este par de cuadrados cuatro veces usando el método horizontalRepeat con el argumento 4, y guardamos el resultado en una variable denominada fila.
Después, modificamos fila usando el método up para colocar otra fila encima con el color invertido, creando así una fila de 8 cuadrados con el patrón de ajedrez en dos filas alternadas. Para esto, aplicamos negative a la variable fila y la colocamos encima usando up.
Finalmente, utilizamos el método verticalRepeat con el argumento 2 para repetir esta estructura dos veces hacia arriba, creando un tablero de 8 x 4, y usamos la función draw para visualizar el resultado.

```
1  #Importar
2  from chessPictures import *
3  from interpreter import draw
4
5  casilleroX2 = square.negative().join(square)
6  fila = casilleroX2.horizontalRepeat(4)
7  fila = fila.up(fila.negative())
8  draw(fila.verticalRepeat(2))
```

Figure 21: Codigo - Ejercicio 6

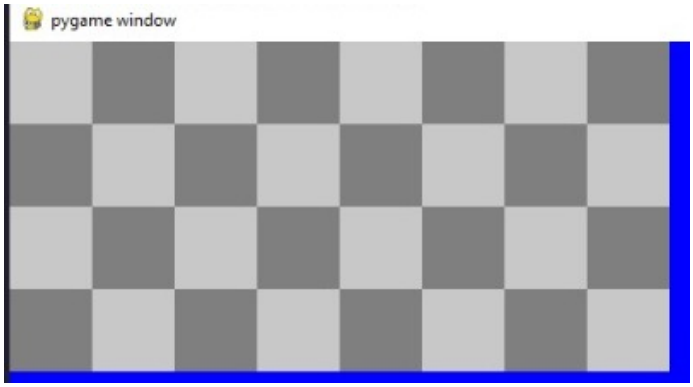

Figure 22: Draw - Ejercicio 6

7mo Ejercicio

Para el último ejercicio, debemos recrear un tablero de ajedrez con las piezas negras en la parte superior. Primero, crearemos la posición de las figuras blancas utilizando diversas variables como cTorre, cCaballo, cAlfil, entre otras, siguiendo el orden respectivo de torre, caballo, alfil, etc. Utilizaremos el método under para colocar cada figura sobre el casillero correspondiente, alternando el color de los casilleros con el método negative.
Por ejemplo, empezaremos con un casillero negro para la torre. Utilizaremos la variable cTorre, en la cual tendremos un casillero negro (square con negative), y aplicaremos el método under con el objeto

rook (que representa la torre). Para el caballo, que sigue a la torre, usaremos la variable cCaballo, utilizando un casillero blanco (square) con el método under llamando al objeto knight (caballo), y así sucesivamente.
Para los peones, haremos la agrupación de dos. En la variable cPeon, primero colocaremos un peón sobre un casillero blanco (square con under y el objeto pawn). Luego, usaremos el método join para colocar a su lado otro peón sobre un casillero negro (square con negative, under con el objeto pawn).
Como ya tenemos las figuras correspondientes en sus casillas, empezaremos a construir el tablero. Primero, crearemos la fila de los peones. Utilizaremos la variable cPeon, la cual contendrá los peones alternando los colores de las casillas. Aplicamos el método horizontalRepeat 4 veces para formar una fila completa de 8 peones, y almacenamos el resultado en la variable filaPeones.
Luego, construiremos la fila de las piezas mediante la variable filaPiezas. Usaremos el método join para unir las piezas en el orden correspondiente: torre, caballo, alfil, reina, rey, alfil, caballo y torre.
A continuación, crearemos la parte media del tablero donde no hay figuras. Para esto, almacenamos dos casilleros alternados en una variable casilleroX2, utilizando el objeto square y el método join con el argumento square llamando al método negative. Después, usaremos casilleroX2 con el método horizontalRepeat 4 veces para crear una fila de 8 casilleros, almacenando el resultado en la variable fila.
Luego, en una variable filaX2, pondremos una fila sobre otra, teniendo en cuenta que al llamar al método up, el argumento debe llamar al método negative. Esto crea dos filas alternadas. Para completar la parte media del tablero, repetiremos filaX2 verticalmente 2 veces utilizando el método verticalRepeat, y almacenamos el resultado en la variable tM.
Finalmente, construiremos el tablero completo usando las variables creadas. Utilizaremos el método up para apilar las filas en el orden adecuado y el método negative para invertir los colores donde sea necesario. Almacenamos el resultado en la variable tablero y usamos la función draw para visualizarlo.

```
2  from chessPictures import *
3  from interpreter import draw
4
5  cTorre = square.negative().under(rock)
6  cCaballo = square.under(knight)
7  cAlfil = square.negative().under(bishop)
8  cReyna = square.under(queen)
9  cRey = square.negative().under(king)
10 cAfil2 = square.under(bishop)
11 cCaballo2 = square.negative().under(knight)
12 cTorre2 = square.under(rock)
13 cPeon = square.under(pawn).join(square.negative().under(pawn))
14 #Fila Peones
15 filaPeones = cPeon.horizontalRepeat(4)
16 #Fila Piezas
17 filaPiezas = cTorre.join(cCaballo).join(cAlfil).join(cReyna).join(cRey).join(cAfil2).join(cCaballo2).join(cTorre2)
18 #Fila medio
19 casillerox2 = square.negative().join(square)
20 fila = casillerox2.horizontalRepeat(4)
21 filax2 = fila.up(fila.negative())
22 tM = filax2.verticalRepeat(2)
23 #Tablero final
24 tablero = filaPiezas.up(filaPeones).up(tM).up(filaPeones.negative()).up(filaPiezas.negative())
25
26 draw(tablero)
```

Figure 23: Codigo - Ejercicio 7

Figure 24: Draw - Ejercicio 7