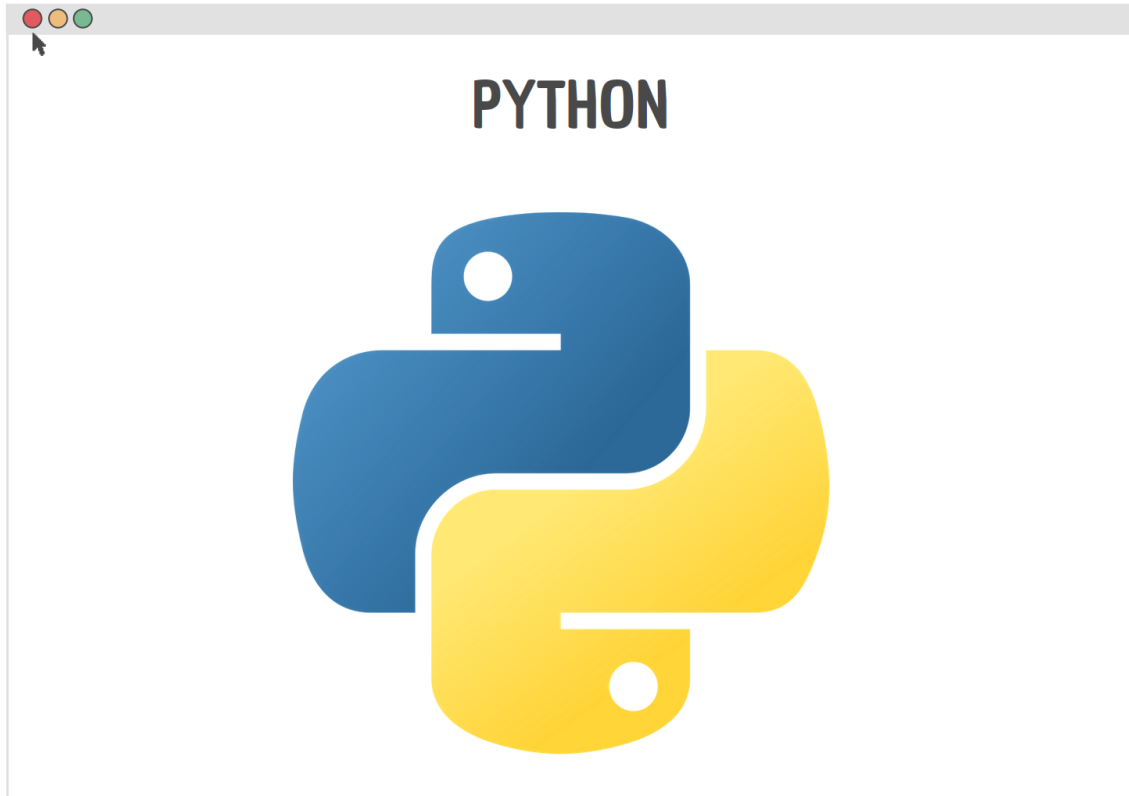




PROGRAMACIÓN WEB 2



Profesor(a):

Carlo Jose Luis Corrales Delgado

Estudiante:

Jorge Luis Mamani Huarsaya

Email:

jmamanihuars@unsa.edu.pe

Repositorio GitHub:

<https://github.com/jorghee/chess-with-python>

24 de mayo, 2024

Implementando los métodos de la clase Picture

Para entender toda la lógica que se explicará a continuación debemos tener claro los siguiente conceptos:

- La clase Picture tiene un campo denominado **img**, el cual es una lista de cadenas de texto (strings). Lo curioso es que esta lista de strings representa la figura que usando el paquete **pygame** y la lógica del archivo `interpreter.py` se transforma de caracteres a pixeles.
- Nosotros no debemos preocuparnos por la conversión de caracteres a pixeles, nuestra aventura es implementar la manipulación de los strings, por lo tanto vamos a trabajar con la lista de strings `self.img`

El método `horizontalMirror()`

Este método devuelve el espejo horizontal de la imagen, es decir, el giro se hace alrededor de una línea imaginaria horizontal trazada en medio de la imagen.

La lógica en mente es iterar sobre la lista de string del objeto Picture al cual se le esta aplicando este método y pasar el último string de la lista al primer elemento de una nueva lista que denominaremos `horizontal`

- Primero creamos la lista `horizontal` vacía con el objetivo de que según la iteración se vayan agregando los strings de `self.img`
- Ahora declaramos la estructura de control `for in` para iterar sobre `self.img`. Empezamos iterando desde el último elemento de esta lista y dicho valor lo asignamos al primer elemento de la lista creada inicialmente vacía.

```
1 # Iterando desde el último elemento de la listas hacia el primer elemento
2 def horizontalMirror(self):
3     size = len(self.img)
4     horizontal = []
5     for value in range(size):
6         horizontal.append(self.img[size - 1 - value])
7     return Picture(horizontal)
```

El método `negative()`

Este método se encarga de cambiar algunos caracteres que se muestran en el script `colors.py` justamente en el diccionario `inverter`.

Para implementar esta funcionalidad rápidamente pensamos en que debemos iterar por cada string de la lista `self.img` y luego iterar por cada caracter y aplicar el método `_invColor()` que se muestra a continuación:

```
1 def \_invColor(self, color):
2     if color not in inverter:
3         return color
4     return inverter[color]
```

Este método mostrado recibe el caracter como argumento y lo pasa como clave al diccionario `mintpythoninverter` el cual devuelve como valor el caracter de cambio.

Con esta información procedemos a codear las siguientes instrucciones.

- Creamos una variable `empty` que contiene un string vacío y creamos una lista `inverter` vacía que regresará la lista nueva.
- Utilizamos un `for in` para iterar sobre la imagen y luego en cada string aplicamos el concepto de **Comprensión de listas** que es sintaxis propia de python. Esta sintaxis nos permite crear una nueva lista de acuerdo a un elemento iterable, en este caso va a ser el string actualmente iterado por el `for in` externo.
- Aquí llamamos al método `_invColor()` que va a recibir los caracteres del string en iteración y aplicará el cambio si es necesario.
- Por último, lo que se ha generado es una lista, pero nosotros deseamos un string. Por lo tanto aplicamos el método `join()` para concatenar todos los caracteres de esta nueva lista generada y ya como string usamos la función `append()` para agregar a la nueva lista que se retornará.

```
1 # Usamos el método _invColor() para cambiar el caracter
2 def negative(self):
3     """ Devuelve un negativo de la imagen """
4     empty = ""
5     inverter = []
6     for value in self.img:
7         inverter.append(empty.join([self._invColor(caracter) for caracter in value]))
8     return Picture(inverter)
```

El método `join()`

Este método debe agregar a la derecha la imagen que recibe como argumento, como nos imaginamos este método es sencillo de implementar porque simplemente necesitamos concatenar a la derecha de la imagen actual.

- Creamos la lista `join` vacía que representará la nueva imagen formada.
- Iniciamos la intrucción de control `for in` para iterar en un rango igual a la longitud de `self.img`
- Luego con la variable `i` que recibe los numeros del rango generado, accedemos al string de la imagen actual y la imagen como argumento finalizando con la concatenación de estos.

```
1 # Usamos el operador '+' para concatenar los strings
2 def join(self, p):
3     """ Devuelve una nueva figura poniendo la figura del argumento
4         al lado derecho de la figura actual """
5     join = []
6     for i in range(len(self.img)):
7         join.append(self.img[i] + p.img[i])
8     return Picture(join)
```

El método `up()`

El método debe poner a `self.img` por encima de la imagen que recibe como argumento. Este se torna sencillo cuando usamos el **Operador de deestructuración** propio de la sintaxis de python.

Lo que el operador de deestructuración es literalmente deestructurar por ejemplo una lista y pasar los elementos de esta como elementos sueltos, no como una lista como es común. Entonces esta sintaxis nos ahorra la necesidad de estar iterando por cada string y recién concatenar.

```
1 # Usamos los operadores de deestructuración propios de la sintaxis de python
2 def up(self, p):
3     up = [*self.img, *p.img]
4     return Picture(up)
```

El método `under()`

Consiste en sobreponer la imagen que se recibe como argumentos sobre la imagen actual. Para esta lógica hemos decidido que el caracter vacío sea el punto de cambio. Es decir, que si algún carácter que conforma cada string de la lista que representa a la figura como argumento es vacío, se toma en cuenta el carácter de la figura actual descartando el carácter de la figura como argumento.

Dificultad : La dificultad fue que el método funcione para sobreponer figuras de diferentes tamaños: cantidad de strings en cada lista y cantidad de caracteres en cada string.

Para solucionar este reto, se tenía la obligación de establecer el tamaño más grande tanto en la cantidad de string como en la cantidad de caracteres. Y de acuerdo a estos límites realizamos las iteraciones y verificaciones para que al final se rellenen con los string y/o caracteres restantes de la figura más grande.

- Primero tenemos que encontrar a la figura con mayor tamaño, es decir con mayor número de strings. Además debemos encontrar el número de caracteres de la figura que contiene strings con mayores caracteres
- Luego, declaramos una lista vacía denominada `under` que representa la lista nueva que se va a generar.
- Empezamos iterando en un rango de la figura más grande. Aquí debemos verificar solo una cosa sencilla, que la variable de control no sobrepase el tamaño de la figura pequeña. En caso sobrepase, lo que queda son los string de la figura más grande, por lo tanto solo agregamos estos strings a la lista `under`.
- Hasta el momento ya hemos solucionado el problema de figuras con cantidad de strings diferentes, ahora necesitamos solucionar las figuras con cantidad de caracteres de cada string diferentes. Para ello hemos creado el método `newString()` que recibe como argumentos al string de la figura actual, el string de la figura como argumento y el número que indica el número de caracteres de la figura con strings más grandes.

```
1 # Implementación que funciona para cualquier longitud de las figuras
2 def under(self, p):
3     """ Devuelve una nueva figura poniendo la figura p sobre la
4         figura actual """
5
6     # Verificamos qué figura tienes más cadenas de texto
7     lower = self.img
8     high = p.img
9     highC = len(p.img[0])
10
11     if len(lower) > len(high):
12         lower = p.img
13         high = self.img
14
15     if len(self.img[0]) > highC:
16         highC = len(self.img[0])
17
```

```
18 # Iteramos sobre las cadenas
19 under = []
20 for i in range(len(high)):
21     if i < len(lower):
22         under.append(self.newString(self.img[i], p.img[i], highC))
23     else:
24         under.append(high[i])
25 return Picture(under)
```

El método newString() usado por under()

Usamos la misma lógica de rellenar con los strings restantes de la figura más grande, solo que ahora son los caracteres del string más grande.

- Creamos una variable **string** que será la que se concatenará con los diferentes caracteres y se retornará en este método.
- Iniciamos el bucle en un rango de **h**, argumento que representa el tamaño del string de la figura más grande. Luego necesitamos realizar varias verificaciones comenzando con saber cuál es la figura que es más grande, parece absurdo hacer esto ahora pero es necesario ya que no se podía pasar directamente con **a** la figura con mayor número de caracteres porque de esta manera se perdía el conocimiento de qué figura es la que debe superponer a la otra. Por ejemplo nosotros sabemos que **p** debe superponer a **s** y eso no lo podemos perder.
- Ahora empezamos con las verificaciones de acuerdo al carácter vacío. Como sabemos que **p** es el string que debe superponer a **s** y sabiendo que **p** es el string con mayor tamaño, debemos verificar que la variable de control **i** no sea mayor al tamaño de string **s** y también verificar que **p[i]** sea igual al carácter vacío. De cumplirse estas 2 condiciones entonces se concatena dicho carácter en iteración, de lo contrario se concatenará el carácter **p[i]**.
- Si es el caso de que **p** no es el string de mayor tamaño entonces se asume que **s** lo es, por lo tanto debemos verificar que **i** sea menor que el tamaño que **p** que ahora es el string pequeño y también verificar que **p[i]** sea diferente del carácter vacío, ya que de lo contrario se concatena con **[s[i]]**.

```
1 # Funcion que itera por cada caracter
2 def newString(self, s, p, h):
3     string = ""
4     for i in range(h):
5         if len(p) == h:
6             if i < len(s) and p[i] == " ":
7                 string += s[i]
8             else:
9                 string += p[i]
10        elif i < len(p) and p[i] != " ":
11            string += p[i]
12        else:
13            string += s[i]
14    return string
```

El método horizontalRepeat()

En este método vamos a utilizar el concepto de **Operador de multiplicación** que se puede aplicar a listas y strings en python. Básicamente el método debe tener la funcionalidad de repetir horizontalmente **n** veces a la figura actual.

- Creamos una lista **horizontal** en la cual se le ira agregando cada string multiplicado.
- Empezamos iterando en la figura actual, esta lógica es sencilla porque simplemente usamos la función **append** a la cual pasamos como argumento el string actualmente iterado multiplicado n veces, lo que hace que dicho string se repita n veces.

```
1 # Usando el operador multiplicador '*' para generar el elemento cuantas veces
  queremos
2 def horizontalRepeat(self, n):
3     """ Devuelve una nueva figura repitiendo la figura actual al costado
4         la cantidad de veces que indique el valor de n """
5     horizontal = []
6     for value in self.img:
7         horizontal.append(value * n)
8         # print(value * n)
9     return Picture(horizontal)
```

El método `verticalRepeat()`

Este método repite también n veces la figura actual pero ahora lo hacemos verticalmente. En este caso necesitamos multiplicar el número de string en la lista, ya no los caracteres. El concepto de **Operador de deestructuración** nuevamente nos simplifica enormemente esta funcionalidad.

- Creamos una lista **vertical**, luego iteramos en un rango de n que es el argumento que el usuario ingresa.
- En la iteración nosotros modificamos la **vertical** que recibe los elementos deestructurados de las listas **vertical** y **self.img**. Aclarar que en la primera iteración **vertical** es una lista vacía y poco a poco se va modificando sumandos sus propios string con los strings de **self.img**.

```
1 # Usando deestructuración de listas
2 def verticalRepeat(self, n):
3     vertical = []
4     for value in range(n):
5         vertical = [*vertical, *self.img]
6     return Picture(vertical)
```

El método `rotate()`

Si no fuera por el método **under()**, este método a implementar hubiera sido el más difícil porque nos pide devolver la figura actual rotada 90 grados. Es necesario mencionar que esta implementación solo funciona para figuras con tamaños y con tamaños de strings iguales.

Para iniciar esta lógica se ha decidido que la rotación va a ser antihoraria. Entonces, la idea básica es que el primer caracter del primer string de la figura se convierta en el primer caracter del último string de la figura, luego que el segundo caracter del primer string de la figura se convierta en el primer caracter del penúltimo string, y así sucesivamente.

- Comenzamos creando una lista **rotate** de un tamaño de **len(self.img)** que contienen strings vacíos. Estos strings vacíos poco a poco se iran concatenando con los caracteres respectivos según la lógica ya explicada.

- Ahora necesitamos iterar por los caracteres de cada string, por lo tanto usamos un **for in** anidado. Entonces cada caracter que iteramos lo ponemos al final de **rotate** pero avanzando hacia el primer string de la lista. Aquí nos ayuda la variable de control **i** que aumenta según el **for in** interno y se reinicia cuando termina dicho bucle.

```
1  # Extra: Sólo para realmente viciosos
2  def rotate(self):
3      """Devuelve una figura rotada en 90 grados, puede ser en sentido horario
4      o antihorario"""
5      rotate = []
6
7      # Creamos la lista vacia pero con n elementos
8      for i in range(len(self.img)):
9          rotate.append("")
10
11     i = 0
12     for value in self.img:
13         for caracter in value:
14             rotate[len(self.img) - 1 - i] += caracter
15             i += 1
16         i = 0
17
18     return Picture(rotate)
```

Resolución de los ejercicios