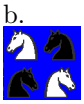
Informe de Laboratorio 04

INFORMACIÓN BÁSICA					
ASIGNATURA:	PROGRAMACION WEB 2				
TITULO DE LA PRÁCTICA:	PYTHON				
NÚMERO DE PRÁCTICA:	04	AÑO LECTIVO:	2023-A	NRO. SEMESTRE:	III
FECHA DE PRESENTACIÓN	08/06/23	HORA DE PRESENTACIÓN:	00:00		
INTEGRANTE (s): - VLADIMIR ARTURO SULLA QUISPE				NOTA (0-20)	
GITHUB :https://github.com/Vladimir2003-debug/PW2-LAB04.git					
DOCENTE(s): - CARLO JOSE LUIS CORRALES DELGADO					

1 TAREA

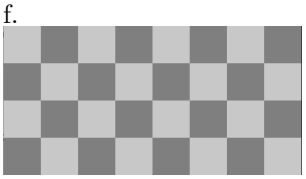
Implemente los métodos de la clase Picture. Se recomienda que implemente la clase picture por etapas, probando realizar los dibujos que se muestran en la siguiente preguntas.
Usando únicamente los métodos de los objetos de la clase Picture dibuje las siguientes figuras (invoque a draw):

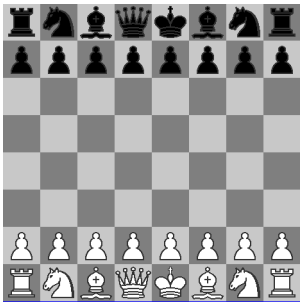
- a.

- b.

- c.

- d.

- e.

- f.

- g.



2 MATERIALES USADOS

- Sistema Operativo Windows 10 Home Single Language
- Python 3.10.4
- virtualenv 20.14.1
- pygame 2.4.0
- Cuenta de GitHub

3 ORGANIZACION Y ESTRUCTURA DEL LABORATORIO

- El contenido que se entrega en este laboratorio es el siguiente:

```
PW2 LAB04/
    |--- __pycache__
    |--- env/
    |--- img
        |--- logo_abet.png
        |--- logo_episunsa.png
        |--- logo_unsa.jpg
    |--- .gitignore
    |--- chessPictures.py
    |--- colors.py
    |--- Ejercicio2a.py
    |--- Ejercicio2b.py
    |--- Ejercicio2c.py
    |--- Ejercicio2d.py
    |--- Ejercicio2e.py
    |--- Ejercicio2f.py
    |--- Ejercicio2g.py
    |--- picture.py
    |--- pieces.py
    |--- README.md
    |--- requirements.py
    |--- Test.py
```

4 SOLUCION

- Primero Solucionamos los metodos de la clase picture. Usamos la figura knight para evaluar si se mueve a la izquierda o la derecha

Listing 1: picture.py

```
1  from colors import *
2  class Picture:
3    def __init__(self, img):
4      self.img = img
5
6    def __eq__(self, other):
7      return self.img == other.img
8
9    def _invColor(self, color):
10     if color not in inverter:
11       return color
12     return inverter[color]
13
14   def verticalMirror(self):
15     """ Devuelve el espejo vertical de la imagen """
16     vertical = []                        # Creacion de vertical
17     for value in self.img:
18         vertical.append(value[::-1])     # Cada linea de la imagen es volteada
19     return Picture(vertical)             # Se retorna objeto Picture
20
21   def horizontalMirror(self):
22     """ Devuelve el espejo horizontal de la imagen """
23     horizontal = self.img[::-1]          # En vez de la linea se voltea los element
24                                          # de la lista de lineas de la imagen
25     return Picture(horizontal)
26
27   def negative(self):
28     """ Devuelve un negativo de la imagen """
29     negative = []                        # Se crea negative
30
31     for line in self.img:
32       newLine = ""                       # Se reemplaza cada punto de la imagen
33       for i in line:                     # Con su negativo en color
34         newLine += self._invColor(i)
35       negative.append(newLine)           # Se agrega la linea creada al negative
36     return Picture(negative)
37
38   def join(self, p):
39     """ Devuelve una nueva figura poniendo la figura del argumento
40         al lado derecho de la figura actual """
41     joinImg = []                         #
42
43     for line in range(len(self.img)): # Concatenamos cada linea de la imagen con
44       joinImg.append(self.img[line]+p.img[line]) # linea de P
45
46     return Picture(joinImg)
```

```
47
48    def up(self, p):
49      """Devuelve una nueva figura poniendo la figura p debajo de
50         la figura actual """
51      upImg = self.img + p.img        # juntamos ambas listas de lineas
52
53      return Picture(upImg)
54
55    def under(self, p):
56      """ Devuelve una nueva figura poniendo la figura p sobre la
57          figura actual """
58
59      length = len(self.img)
60
61      underImg = []
62
63      for line in range(length):        #
64        item = ""
65        for i in range(length):
66          if p.img[line][i] == " ":     # Cada punto blanco en la imagen p
67            item += self.img[line][i]   # Es reemplazada por la pocicion en
68          else:                         # la imagen self.img
69            item += p.img[line][i]
70        underImg.append(item)
71
72      return Picture(underImg)
73
74    def horizontalRepeat(self, n):
75      """ Devuelve una nueva figura repitiendo la figura actual al costado
76          la cantidad de veces que indique el valor de n """
77      repeat = []
78      for line in self.img:
79        repeat.append(line*n)           # Multiplicamos cada linea de self.img por
80      return Picture(repeat)
81
82    def verticalRepeat(self, n):
83      repeat = self.img * n             # Multiplicamos la imagen por n veces
84      return Picture(repeat)
85
86    #Extra: S lo para realmente viciosos
87    def rotate(self):
88      """Devuelve una figura rotada en 90 grados, puede ser en sentido horario
89      o antihorario"""
90
91      length = len(self.img)            # Extremos el numero de linas
92      rotateImg = []
93
94      for i in range(length):
95        rotateLine = ""                 # creamos una linea
96        for line in self.img:
97          rotateLine += line[i]         # cada linea va a ser un string de caracte
98        rotateImg.append(rotateLine)    # i de cada linea de la imagen
```

```
99
100     return Picture(rotateImg)
```

- Luego solo es cuestion de resolver los ejercicios jugando con los metodos picture
- Ejercicio2a.py

Listing 2: Ejercicio2a.py

```
1 from interpreter import draw
2 from chessPictures import *
3
4 # dibujamos knight junto a su negativo lo pocicinamos encima de su contraparte
5 draw(knight.join(knight.negative()).up(knight.negative().join(knight)))
```

- Ejercicio2b.py

Listing 3: Ejercicio2b.py

```
1 from interpreter import draw
2 from chessPictures import *
3
4 # Hacemos casi lo mismo que el Ejercicio 2b pero a la parte de abajo lo giramos
5 draw(knight.join(knight.negative()).up(knight.join(knight.negative()).verticalMi
```

- Ejercicio2c.py

Listing 4: Ejercicio2c.py

```
1 from interpreter import draw
2 from chessPictures import *
3
4 #Simplemente repetimos a la Queen
5 draw(queen.horizontalRepeat(4))
```

- Ejercicio2d.py

Listing 5: Ejercicio2d.py

```
1 from interpreter import draw
2 from chessPictures import *
3
4 # Juntamos un cuadrado con su negativo y lo repetimos 4 veces
5 draw(square.join(square.negative()).horizontalRepeat(4))
```

- Ejercicio2e.py

Listing 6: Ejercicio2e.py

```
1 from interpreter import draw
2 from chessPictures import *
3
4 # Lo mismo que el ejercicio 2d pero al revez
5 draw(square.join(square.negative()).horizontalRepeat(4))
```

- Ejercicio2f.py

Listing 7: Ejercicio2f.py

```
from interpreter import draw
from chessPictures import *

# Dibujamos una linea de cuadrados
line = square.join(square.negative()).horizontalRepeat(4)
# Luego lo ponemos encima de su negativo y lo repetimos 2 veces
draw(line.up(line.negative()).verticalRepeat(2))
```

- Ejercicio2g.py

Listing 8: Ejercicio2g.py

```
from interpreter import draw
from chessPictures import *

# Images adicionales
sqRock = square.negative().under(rock)
sqKnight = square.under(knight)
sqBishop = square.negative().under(bishop)
sqQueen = square.under(queen)
sqKing = square.negative().under(king)
sqBishop2 = square.under(bishop)
sqKnight2 = square.negative().under(knight)
sqRock2 = square.under(rock)

# Esta linea de piezas se ubicaran en la parte inferior
linePieces = sqRock.join(sqKnight).join(sqBishop).join(sqQueen).join(sqKing).joi
# Una linea de peones
linePawns = square.under(pawn).join(square.negative().under(pawn)).horizontalRep
# La tabla que va al medio de la tabla sin piezas
lineTable = square.join(square.negative()).horizontalRepeat(4)
table = lineTable.up(lineTable.negative()).verticalRepeat(2)
# La parte de las negras es el negativo de la parte blanca tanto linea de peones
# todo se junta y se muestra
draw(linePieces.negative().img + linePawns.negative().img + table.img + linePawn
```

5 CUESTIONARIO

- ¿Qué son los archivos *.pyc?
  Los archivos PYC son utilizados por el lenguaje de programación Python. PYC es un archivo ejecutable que contiene el código de bytes compilado para un programa escrito en Python. Bytecode es un conjunto de instrucciones para que el intérprete ejecute el programa. Basicamente son el archivo .py pero en version de codigos de bytes

- ¿Para qué sirve el directorio pycache?
  Cuando el interprete de python ejecuta archivos .py lo compila en codigo de bytes y lo almacena en el directorio pycache con extension .pyc para que a la siguiente que se ejecute el archivo no tenga que volver a hacer el mismo procedemiento.

- ¿Cuáles son los usos y lo que representa el subguión en Python?
  El guion bajo se usa para evitar conflictos con palabras clave o elementos.

6 REFERENCIAS

- https://www.w3schools.com/python/python$_r ef erence.asp$
- https://docs.python.org/3/tutorial/
- https://www.file-extension.info/es/format/pyc
- https://stackoverflow.com/questions/16869024/what-is-pycache
- https://frankgalandev.com/que-significa-el-guion-bajo-en-python/: :text=Se