Informe de Laboratorio 05

Tema: Pyton

Nota

Estudiante	Escuela	Asignatura
Alexandra Raquel Quispe	Escuela Profesional de Ingeniería de Sistemas	Programación Web II Semestre: I Código: 20231001

Laboratorio	Tema	Duración
05	Pyton	12 horas

Semestre académico	Fecha de inicio	Fecha de entrega
2024 - A	Del 21 Mayo 2024	Al 25 Mayo 2023

1. Ejercicios Propuestos

En esta tarea, individualmente usted pondrá en práctica sus conocimientos de programación en Python para dibujar un tablero de Ajedrez. La parte gráfica ya está programada, usted sólo tendrá que concentrarse en las estructuras de datos subyacentes. Con el código proporcionado usted dispondrá de varios objetos de tipo `Picture` para poder realizar su tarea:

- `rock`
- `knight`
- `bishop`
- `queen`
- `king`
- `square`

Estos objetos estarán disponibles importando la biblioteca: `chessPictures` y estarán internamente representados con arreglos de strings que podrá revisar en el archivo `pieces.py`.

La clase `Picture` tiene un sólo atributo: el arreglo de strings `img`, el cual contendrá la representación en caracteres de la figura que se desea dibujar. La clase `Picture` ya cuenta con una función implementada, no debe modificarla, pero si puede usarla para implementar sus otras funciones:

- `_invColor`: recibe un color como un carácter de texto y devuelve su color negativo, también como texto, deberá revisar el archivo `colors.py` para conocer los valores negativos de cada carácter.

La clase `Picture` contará además con varios métodos que usted deberá implementar:

- `verticalMirror`: Devuelve el espejo vertical de la imagen
- `horizontalMirror`: Devuelve el espejo horizontal de la imagen
- `negative`: Devuelve un negativo de la imagen
- `join`: Devuelve una nueva figura poniendo la figura del argumento al lado derecho de la figura actual
- `up`: Devuelve una nueva figura poniendo la figura recibida como argumento, encima de la figura actual
- `under`: Devuelve una nueva figura poniendo la figura recibida como argumento, sobre la figura actual
- `horizontalRepeat`: Devuelve una nueva figura repitiendo la figura actual al costado la cantidad de veces que indique el valor de `n`
- `verticalRepeat`: Devuelve una nueva figura repitiendo la figura actual debajo, la cantidad de veces que indique el valor de `n`

Tenga en cuenta que para implementar todos estos métodos, sólo deberá trabajar sobre la representación interna de un `Picture`, es decir su atributo `img`.

Para dibujar una objeto `Picture` bastará importar el método `draw` de la biblioteca `interpreter` y usarlo de la siguiente manera:

```
>>> from chessPictures import *
>>> from interpreter import draw
>>> draw(rock)
```

Considere el repositorio: `https://github.com/rescobedoq/pw2/tree/main/labs/lab04/Tarea-del-Ajedrez`

2. Ejercicios

Para resolver los siguientes ejercicios sólo está permitido usar ciclos, condicionales, definición de listas por comprensión, sublistas, `map`, `join`, `+`, `lambda`, `zip`, `append`, `pop`, `range`. Implemente los métodos de la clase `Picture`. Se recomienda que implemente la clase `Picture` por etapas, probando realizar los dibujos que se muestran en las siguientes preguntas. Usando únicamente los métodos de los objetos de la clase `Picture`, dibuje las siguientes figuras (invoque a `draw`):

3. Equipos, materiales y temas utilizados

- **Sistema Operativo:** Ubuntu GNU/Linux 23.04 Lunar Lobster 64 bits, Kernel 6.2.
- **Editor de texto:** VIM 9.0.
- **Entorno de desarrollo:** Python 3.10.
- **Control de versiones:** Git 2.39.2.
- **Repositorio:** Cuenta en GitHub con el correo institucional.
- **Bibliotecas utilizadas:**
  - **chessPictures:** Para el manejo y representación gráfica de piezas de ajedrez.

- **interpreter:** Para dibujar los objetos de tipo `Picture`.

- **Tecnologías y métodos utilizados:**
  - **Python:** Utilizado para implementar las estructuras de datos y métodos necesarios para manipular y visualizar las piezas de ajedrez.
  - **GitHub:** Para la gestión y control de versiones del código fuente del proyecto.

4. URL de Repositorio GitHub

- URL del Repositorio GitHub para clonar o recuperar.
- `https://github.com/aquispearr/Pweb2_Lab05.git`
- URL deL video explicativo.
- `https://docs.google.com/document/d/1eUpm0Yp-VVFlS1BXQTqXK0amFvjJQZWHBrFFK2JonhY/edit?usp=sharing`

5. Actividades con el repositorio GitHub

5.1. Implementación de la clase `Picture`

En esta sección se explica la implementación de la clase `Picture` que permite manipular y dibujar figuras representadas por arreglos de strings. A continuación, se detallan las funciones y métodos de la clase.

- **Definición e inicialización de la clase**

Listing 1: Definición e inicialización de la clase `Picture`

```
from colors import *

class Picture:
    def __init__(self, img):
        self.img = img

    def __eq__(self, other):
        return self.img == other.img

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

Explicación:

- `__init__`: Este método es el constructor de la clase `Picture`, que inicializa un objeto `Picture` con el atributo `img`, que es una lista de strings que representa la imagen.
- `__eq__`: Este método permite comparar dos objetos `Picture` para verificar si son iguales, basándose en su atributo `img`.
- `_invColor`: Este método privado invierte el color de un carácter utilizando un diccionario de colores invertidos.

- **Método** `verticalMirror`

Listing 2: Método `verticalMirror`

```
def verticalMirror(self):
    vertical = []
    for value in self.img:
        vertical.append(value[::-1])
    return Picture(vertical)
```

Explicación: Este método devuelve un nuevo objeto `Picture` cuya imagen es el espejo vertical de la imagen original. Para lograrlo, se invierte cada string de la lista `img` utilizando el slicing de Python `[::-1]`.

- **Método** `horizontalMirror`

Listing 3: Método `horizontalMirror`

```
def horizontalMirror(self):
    return Picture(self.img[::-1])
```

Explicación: Este método devuelve un nuevo objeto `Picture` cuya imagen es el espejo horizontal de la imagen original. Para ello, se invierte la lista `img` utilizando el slicing de Python `[::-1]`.

- **Método** `negative`

Listing 4: Método `negative`

```
def negative(self):
    negativo = []
    for value in self.img:
        filaInvertida = ""
        for caracter in value:
            filaInvertida += self._invColor(caracter)
        negativo.append(filaInvertida)
    return Picture(negativo)
```

Explicación: Este método devuelve un nuevo objeto `Picture` cuya imagen es el negativo de la imagen original. Para lograrlo, se itera sobre cada fila de la imagen (`self.img`) y se crea una nueva fila invirtiendo el color de cada carácter utilizando el método `_invColor`. Todas las filas invertidas se almacenan en una nueva lista llamada `negativo`, y luego se crea una nueva instancia de `Picture` con esta lista de filas invertidas.

- **Método** `join`

Listing 5: Método `join`

```
def join(self, p):
    unido = []
    for i in range(len(self.img)):
        filaJunta = self.img[i] + p.img[i]
        unido.append(filaJunta)
    return Picture(unido)
```

Explicación: Este método devuelve un nuevo objeto `Picture` cuya imagen es la concatenación horizontal de la imagen actual con la imagen del objeto `Picture` pasado como argumento. Se

itera sobre los índices de las filas y se concatenan las filas correspondientes de `self.img` y `p.img`. Las filas concatenadas se almacenan en una nueva lista llamada `unido`, y se crea una nueva instancia de `Picture` con esta lista de filas unidas.

- **Método** `up`

Listing 6: Método `up`

```
def up(self, p):
    compuesto = []
    for fila in self.img:
        compuesto.append(fila)
    for fila in p.img:
        compuesto.append(fila)
    return Picture(compuesto)
```

Explicación: Este método devuelve un nuevo objeto `Picture` cuya imagen es la concatenación vertical de la imagen actual con la imagen del objeto `Picture` pasado como argumento. Se crea una nueva lista llamada `compuesto` que contiene todas las filas de `self.img` seguidas de todas las filas de `p.img`. Esta lista se utiliza para crear una nueva instancia de `Picture` que representa la imagen compuesta.

- **Método** `under`

Listing 7: Método `under`

```
def under(self, p):
    sobrepuesto = []
    for fila_index, fila in enumerate(p.img):
        filaSobrepuesta = ""
        for col_index, caracter in enumerate(fila):
            if caracter != " ":
                filaSobrepuesta += caracter
            else:
                filaSobrepuesta += self.img[fila_index][col_index]
        sobrepuesto.append(filaSobrepuesta)
    return Picture(sobrepuesto)
```

Explicación: Este método devuelve un nuevo objeto `Picture` cuya imagen es la superposición de la imagen del objeto `Picture` pasado como argumento sobre la imagen actual. Se itera sobre las filas y columnas de `p.img` y, si un carácter en `p.img` es un espacio en blanco, se reemplaza por el carácter correspondiente de `self.img`. Las filas superpuestas se almacenan en una nueva lista llamada `sobrepuesto`, y se crea una nueva instancia de `Picture` con esta lista.

- **Método** `horizontalRepeat`

Listing 8: Método `horizontalRepeat`

```
def horizontalRepeat(self, n):
    repetidoH = []
    for fila in self.img:
        filaRepetida = fila * n
        repetidoH.append(filaRepetida)
    return Picture(repetidoH)
```

Explicación: Este método devuelve un nuevo objeto `Picture` cuya imagen es la repetición horizontal de la imagen actual `n` veces. Se itera sobre cada fila de `self.img` y se crea una

nueva fila repitiendo esa fila `n` veces utilizando la multiplicación de cadenas (`fila * n`). Las filas repetidas se almacenan en una nueva lista llamada `repetidoH`, y se crea una nueva instancia de `Picture` con esta lista.

- **Método** `verticalRepeat`

Listing 9: Método `verticalRepeat`

```
def verticalRepeat(self, n):
    repetidoV = []
    for i in range(n):
        for fila in self.img:
            repetidoV.append(fila)
    return Picture(repetidoV)
```

Explicación: Este método devuelve un nuevo objeto `Picture` cuya imagen es la repetición vertical de la imagen actual `n` veces. Se crea una nueva lista llamada `repetidoV` que contiene todas las filas de `self.img` repetidas `n` veces. Esta lista se utiliza para crear una nueva instancia de `Picture` que representa la imagen repetida verticalmente.

- **Método** `rotate`

Listing 10: Método `rotate`

```
def rotate(self):
    b = []
    lenSelf = len(self.img)
    for i in range(lenSelf):
        a = ""
        for value in self.img:
            a += value[lenSelf - 1 - i]
        b.append(a)
    return Picture(b)
```

Explicación: Este método devuelve un nuevo objeto `Picture` cuya imagen es la rotación de 90 grados en sentido antihorario de la imagen original. Se crea una nueva lista llamada `b` que almacenará las filas de la imagen rotada. Se itera sobre los índices de las columnas de la imagen original y, para cada índice `i`, se crea una nueva cadena `a` que contiene los caracteres de la columna `i` de la imagen original, pero en orden inverso (comenzando desde la parte inferior de la columna). Cada cadena `a` se agrega a la lista

5.2. Ejercicios propuestos

5.2.1. Ejercicio 2A

Listing 11: Código del ejercicio 2A

```
from interpreter import draw
from chessPictures import *

caballo1 = Picture(KNIGHT)
caballo2 = caballo1.negative()
fila1 = caballo1.join(caballo2)
fila2 = fila1.negative()
tablero = fila1.up(fila2)
```

```
draw(tablero)
```

Explicación:

- Se importan los módulos necesarios: `interpreter` para la función `draw` y `chessPictures` para acceder a la representación del caballo de ajedrez (`KNIGHT`).
- Se crea un objeto `Picture` `caballo1` a partir de la representación del caballo de ajedrez.
- Se crea un objeto `Picture` `caballo2` que es el negativo de `caballo1`.
- Se concatena horizontalmente `caballo1` y `caballo2` para formar la `fila1`.
- Se crea `fila2` como el negativo de `fila1`.
- Se concatena verticalmente `fila1` y `fila2` para formar el `tablero`.
- Finalmente, se utiliza la función `draw` para dibujar el `tablero`.

Resultado: El resultado de este código es un tablero de ajedrez formado por caballos blancos y negros, como se muestra en la siguiente imagen:

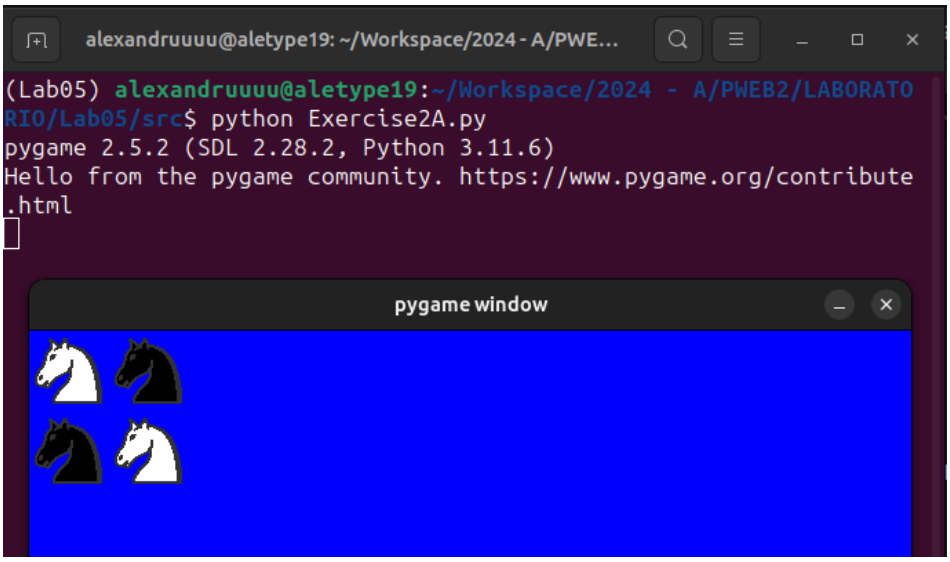

Figura 1: Resultado del ejercicio 2A

5.2.2. Ejercicio 2B

Listing 12: Código del ejercicio 2B

```
from interpreter import draw
from chessPictures import *

caballo1 = Picture(KNIGHT)
caballo2 = caballo1.negative()
fila1 = caballo1.join(caballo2)
fila2 = fila1.verticalMirror()
tablero = fila1.up(fila2)
draw(tablero)
```

Explicación:

- Se importan los módulos necesarios: `interpreter` para la función `draw` y `chessPictures` para acceder a la representación del caballo de ajedrez (`KNIGHT`).
- Se crea un objeto `Picture caballo1` a partir de la representación del caballo de ajedrez.
- Se crea un objeto `Picture caballo2` que es el negativo de `caballo1`.
- Se concatena horizontalmente `caballo1` y `caballo2` para formar la `fila1`.
- Se crea `fila2` como el espejo vertical de `fila1` utilizando el método `verticalMirror`.
- Se concatena verticalmente `fila1` y `fila2` para formar el `tablero`.
- Finalmente, se utiliza la función `draw` para dibujar el `tablero`.

Resultado: El resultado de este código es un tablero de ajedrez formado por caballos blancos y negros, donde la mitad inferior es el espejo vertical de la mitad superior, como se muestra en la siguiente imagen:

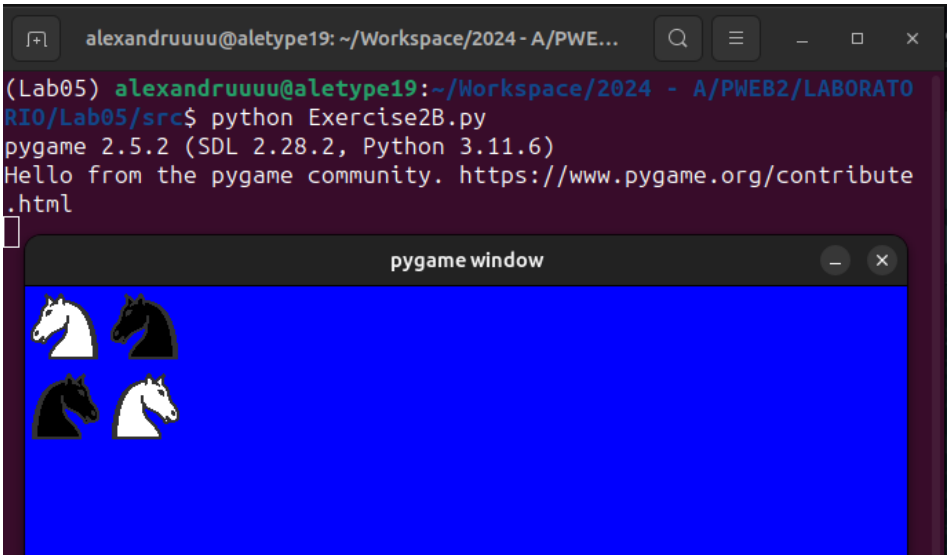

Figura 2: Resultado del ejercicio 2B

5.2.3. Ejercicio 2C

Listing 13: Código del ejercicio 2C

```
from interpreter import draw
from chessPictures import *

reyna = Picture(QUEEN)
tablero = reyna.horizontalRepeat(4)
draw(tablero)
```

Explicación:

- Se crea un objeto `Picture reyna` a partir de la representación de la reina de ajedrez.
- Se utiliza el método `horizontalRepeat` con el argumento `4` para repetir la imagen de la reina horizontalmente cuatro veces, creando un nuevo objeto `Picture` llamado `tablero`.
- Finalmente, se utiliza la función `draw` para dibujar el `tablero`.

Resultado: El resultado de este código es una fila con cuatro representaciones de la reina de ajedrez, como se muestra en la siguiente imagen:

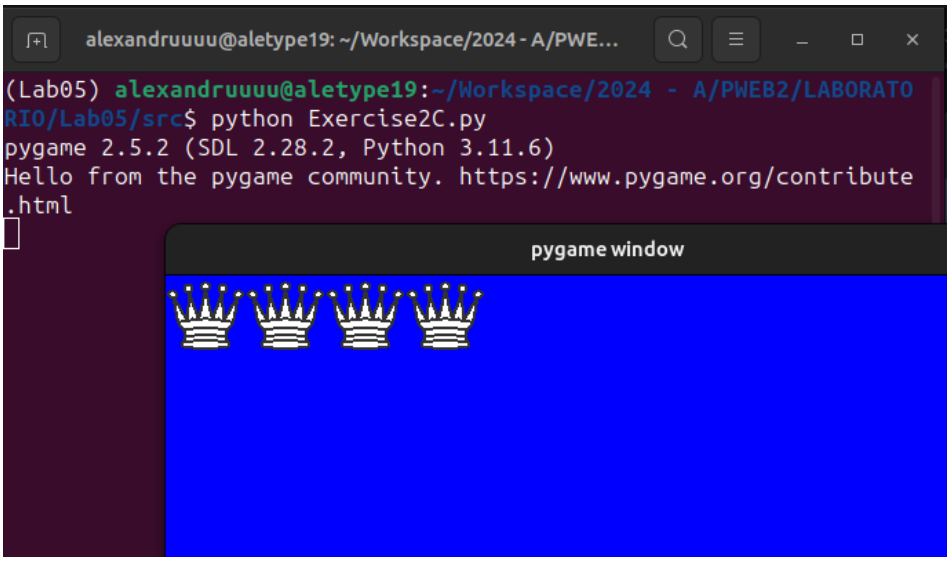

Figura 3: Resultado del ejercicio 2C

5.2.4. Ejercicio 2D

Listing 14: Código del ejercicio 2D

```
from interpreter import draw
from chessPictures import *

cuadroBlanco = Picture(SQUARE)
cuadroNegro = cuadroBlanco.negative()
tablero1 = cuadroBlanco.join(cuadroNegro)
tablero = tablero1.horizontalRepeat(4)
draw(tablero)
```

Explicación:

- Se importan los módulos necesarios.
- Se crea un objeto `cuadroBlanco` a partir de la representación de un cuadro blanco (`SQUARE`).
- Se crea `cuadroNegro` como el negativo de `cuadroBlanco`.
- Se concatenan horizontalmente `cuadroBlanco` y `cuadroNegro` para formar `tablero1`.
- Se repite `tablero1` cuatro veces horizontalmente utilizando `horizontalRepeat` para crear `tablero`.
- Se dibuja `tablero` con la función `draw`.

Resultado: Una fila con ocho cuadros alternando entre blanco y negro, como se muestra:

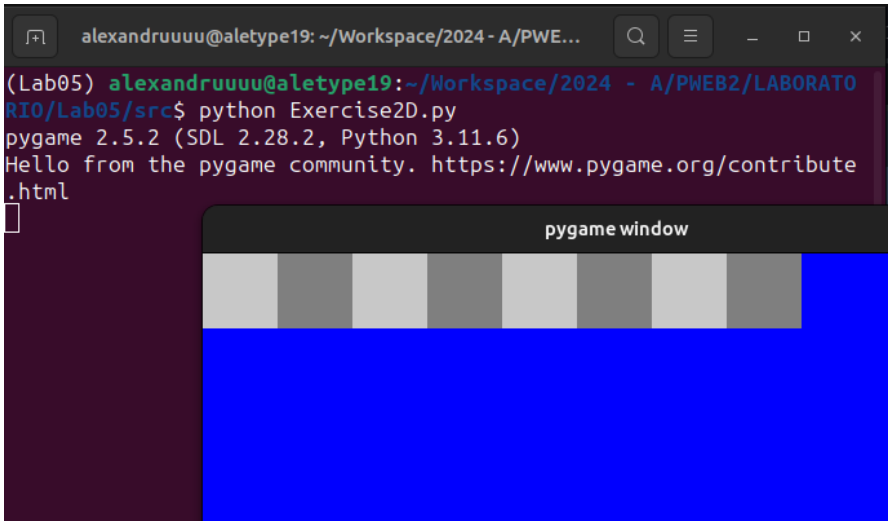

Figura 4: Resultado del ejercicio 2D

5.2.5. Ejercicio 2E

Listing 15: Código del ejercicio 2E

```
from interpreter import draw
from chessPictures import *

cuadroBlanco = Picture(SQUARE)
cuadroNegro = cuadroBlanco.negative()
tablero1 = cuadroNegro.join(cuadroBlanco)
tablero = tablero1.horizontalRepeat(4)
draw(tablero)
```

Explicación: La única diferencia con el ejercicio 2D es que la concatenación horizontal de `cuadroNegro` y `cuadroBlanco` se realiza en orden inverso para formar `tablero1`.

Resultado: Una fila con ocho cuadros alternando entre negro y blanco, como se muestra:

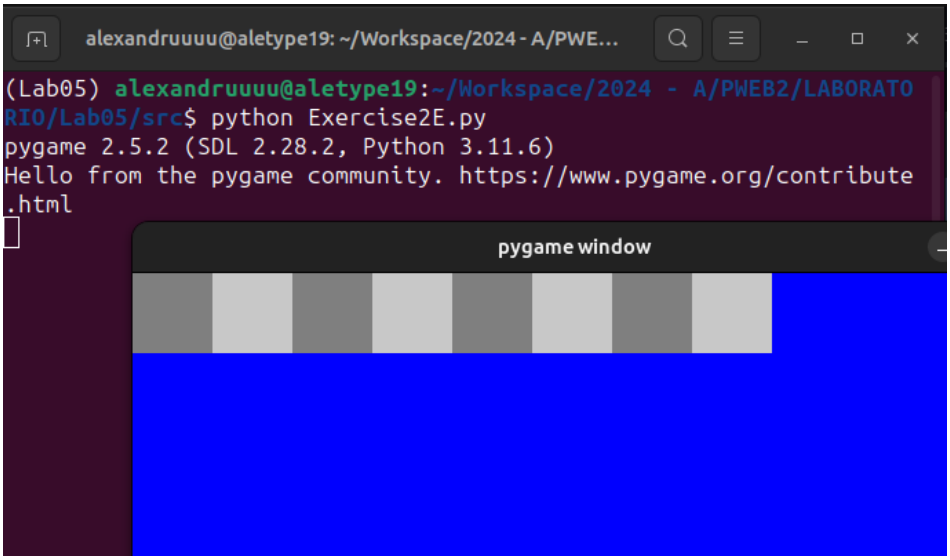

Figura 5: Resultado del ejercicio 2E

5.2.6. Ejercicio 2F

Listing 16: Código del ejercicio 2F

```
from interpreter import draw
from chessPictures import *

cuadroBlanco = Picture(SQUARE)
cuadroNegro = cuadroBlanco.negative()
tablero1 = cuadroBlanco.join(cuadroNegro).horizontalRepeat(4)
tablero2 = tablero1.negative()
tablero = tablero1.up(tablero2).verticalRepeat(2)
draw(tablero)
```

Explicación:

- Se importan los módulos necesarios.
- Se crea un objeto `cuadroBlanco` a partir de la representación de un cuadro blanco (`SQUARE`).
- Se crea `cuadroNegro` como el negativo de `cuadroBlanco`.
- Se concatenan horizontalmente `cuadroBlanco` y `cuadroNegro`, y luego se repite este patrón cuatro veces horizontalmente para formar `tablero1`.
- Se crea `tablero2` como el negativo de `tablero1`.
- Se concatenan verticalmente `tablero1` y `tablero2` para formar una fila completa, y luego se repite este patrón dos veces verticalmente para formar `tablero`.
- Se dibuja `tablero` con la función `draw`.

Resultado: Un tablero de ajedrez completo de 8x8 cuadros alternando entre blanco y negro, como se muestra:

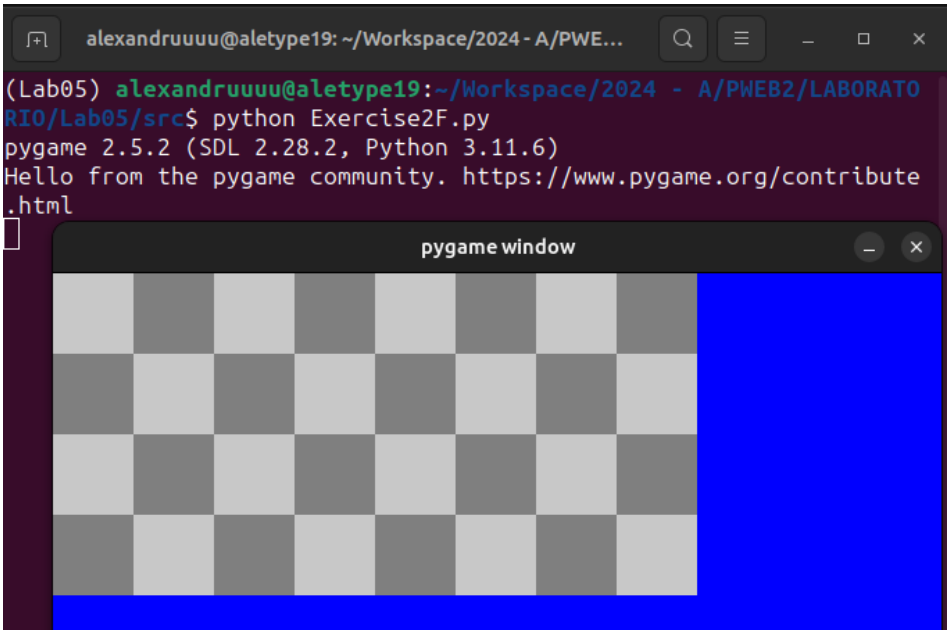

Figura 6: Resultado del ejercicio 2F

5.2.7. Ejercicio Final

Listing 17: Código del ejercicio final

```
from interpreter import draw
from chessPictures import *

cuadroBlanco = Picture(SQUARE)
cuadroNegro = cuadroBlanco.negative()
tablero1 = cuadroBlanco.join(cuadroNegro).horizontalRepeat(4)
tablero2 = tablero1.negative()
fila1 =
    rock.join(knight).join(bishop).join(queen).join(king).join(bishop).join(knight).join(rock)
fila2 = pawn.horizontalRepeat(8)
tableroSuperior = tablero2.up(tablero1).under(fila1.up(fila2)).negative()
tableroMedio = tablero1.up(tablero2).verticalRepeat(2)
tableroInferior = tablero1.up(tablero2).under(fila2.up(fila1))
final = tableroSuperior.up(tableroMedio).up(tableroInferior)
draw(final)
```

Explicación: Este código crea un tablero de ajedrez completo con todas las piezas en sus posiciones iniciales. Primero, se construyen las filas de cuadros blancos y negros utilizando los métodos `join` y `horizontalRepeat`. Luego, se crean las filas con las piezas de ajedrez, utilizando las representaciones importadas desde el módulo `chessPictures`.

A continuación, se construyen tres secciones principales del tablero: la sección superior con las piezas blancas, la sección media con cuadros vacíos y la sección inferior con las piezas negras. Cada sección se crea combinando las filas de cuadros y piezas mediante los métodos `up`, `under` y `verticalRepeat`.

Finalmente, se ensambla el tablero completo concatenando verticalmente las tres secciones prin-

cipales utilizando el método `up`. El resultado final se dibuja con la función `draw`.

Resultado: Un tablero de ajedrez completo con todas las piezas en sus posiciones iniciales, como se muestra en la siguiente imagen:

Figura 7: Resultado del ejercicio final

6. Archivos entregados

A continuación, se muestra la estructura de archivos y directorios entregados para este laboratorio:

Listing 18: Estructura de archivos y directorios

```
(Lab05) alexandruuuu@aletype19:~/Workspace/2024 - A/PWEB2/LABORATORIO/Lab05/src$ tree
.
        chessPictures.py
        colors.py
        Exercise2A.py
        Exercise2B.py
```

```
        Exercise2C.py
        Exercise2D.py
        Exercise2E.py
        Exercise2F.py
        Exercise2G.py
        interpreter.py
 __  latex
_    __  img
_   _           1.png
_   _           2.png
_   _           3.png
_   _           4.png
_   _           5.png
_   _           6.png
_   _           7.png
_   _           logo_abet.png
_   _           logo_episunsa.png
_   _           logo_unsa.jpg
_           InformeLabFinal.tex
_           Pweb2Report_Lab05.pdf
_    __  src
_               horizontal_mirror.py
_               horizontal_repeat.py
_               initialization.py
_               join.py
_               negative.py
_               rotate.py
_               under.py
_               up.py
_               vertical_mirror.py
_               vertical_repeat.py
        picture.py
        pieces.py
        Pweb2Report_Lab05.zip
 __  __pycache__
            chessPictures.cpython-311.pyc
            colors.cpython-311.pyc
            interpreter.cpython-311.pyc
            picture.cpython-311.pyc
            pieces.cpython-311.pyc
```

7. Rubrica

7.1. Rúbrica para el contenido del Informe y demostración

- El alumno debe marcar o dejar en blanco en celdas de la columna **Checklist** si cumplio con el ítem correspondiente.
- Si un alumno supera la fecha de entrega, su calificación será sobre la nota mínima aprobada, siempre y cuando cumpla con todos lo items.
- El alumno debe autocalificarse en la columna **Estudiante** de acuerdo a la siguiente tabla:

Tabla 1: Niveles de desempeño

	Nivel			
Puntos	Insatisfactorio 25 %	En Proceso 50 %	Satisfactorio 75 %	Sobresaliente 100 %
2.0	0.5	1.0	1.5	2.0
4.0	1.0	2.0	3.0	4.0

Tabla 2: Rúbrica para contenido del Informe y demostración

Contenido y demostración		Puntos	Checklist	Estudiante	Profesor
1. GitHub	Hay enlace URL activo del directorio para el laboratorio hacia su repositorio GitHub con código fuente terminado y fácil de revisar.	2	X	2	
2. Commits	Hay capturas de pantalla de los commits más importantes con sus explicaciones detalladas. (El profesor puede preguntar para refrendar calificación).	4	X	2	
3. Código fuente	Hay porciones de código fuente importantes con numeración y explicaciones detalladas de sus funciones.	2	X	1	
4. Ejecución	Se incluyen ejecuciones/pruebas del código fuente explicadas gradualmente.	2	X	1	
5. Pregunta	Se responde con completitud a la pregunta formulada en la tarea. (El profesor puede preguntar para refrendar calificación).	2	X	2	
6. Fechas	Las fechas de modificación del código fuente estan dentro de los plazos de fecha de entrega establecidos.	2	X	2	
7. Ortografía	El documento no muestra errores ortográficos.	2	X	2	
8. Madurez	El Informe muestra de manera general una evolución de la madurez del código fuente, explicaciones puntuales pero precisas y un acabado impecable. (El profesor puede preguntar para refrendar calificación).	4	X	3	
Total		20		15	