

# Informe de Laboratorio 05

## Tema: Python

Nota

Estudiante	Escuela	Asignatura
Victor Gonzalo Maldonado Vilca vmaldonadov@unsa.edu.pe	Escuela Profesional de Ingeniería de Sistemas	Programación Web 2 Semestre: III Código: 1702122

Tarea	Tema	Duración
05	Python	2 horas

Semestre académico	Fecha de inicio	Fecha de entrega
2024 - A	Del 9 de abril de 2024	Al 25 de mayo de 2024

## 1. Introducción

### 1.1. Python

Lenguaje de programación de alto nivel, interpretado y de propósito general. Creado por Guido van Rossum y lanzado por primera vez en 1991. Conocido por sintaxis clara y legible, lo que facilita su aprendizaje. Python soporta múltiples paradigmas de programación, incluyendo la programación orientada a objetos, la programación funcional y la programación imperativa.

### 1.2. Pip

Es el gestor de paquetes oficial para Python. Permite instalar, actualizar y desinstalar paquetes de software escritos en Python. pip facilita la instalación de bibliotecas y paquetes que no están incluidos en la biblioteca estándar de Python, y es especialmente útil para manejar dependencias en proyectos.

### 1.3. Pygame

Conjunto de módulos en Python diseñados para escribir videojuegos. Pygame permite la creación de gráficos 2D y el manejo de eventos, sonido y controladores de entrada (como el teclado y el ratón).

### 1.4. Virtual Environment

Es una herramienta para crear entornos virtuales en Python, aislando las dependencias de proyectos diferentes. Permite usar distintas versiones de paquetes y de Python sin conflictos.

## 2. Objetivos

- Practicar los principios de programación usando Python
- Mostrar un ejemplo de separación de intereses en clases: el modelo (lista de strings) de su vista (dibujo de gráficos).

## 3. Tarea

### 3.1. 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:





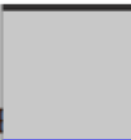
	rock
	knight
	bishop
	queen
	king
	square

Figura 1: Figuras Ajedrez

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 textbfPicture contará además con varios métodos que usted deberá implementar:

1. **verticalMirror:** Devuelve el espejo vertical de la imagen.
2. **horizontalMirror:** Devuelve el espejo horizontal de la imagen.
3. **negative:** Devuelve un negativo de la imagen.
4. **join:** Devuelve una nueva figura poniendo la figura del argumento al lado derecho de la figura actual.
5. **up:** Devuelve una nueva figura poniendo la figura recibida como argumento, encima de la figura actual.
6. **under:** Devuelve una nueva figura poniendo la figura recibida como argumento, sobre la figura actual.
7. **horizontalRepeat:** Devuelve una nueva figura repitiendo la figura actual al costado la cantidad de veces que indique el valor de n.
8. **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)
```



Figura 2: Forma de graficar

### 3.2. 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.**

1. 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.
2. Usando únicamente los métodos de los objetos de la clase Picture dibuje las siguientes figuras (invoque a draw):

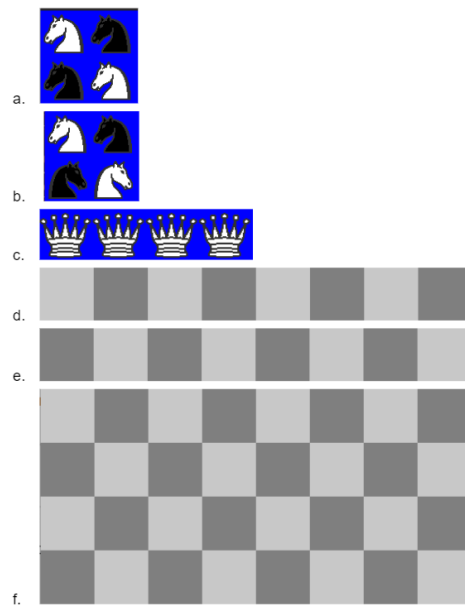


Figura 3: Ejercicios parte 1

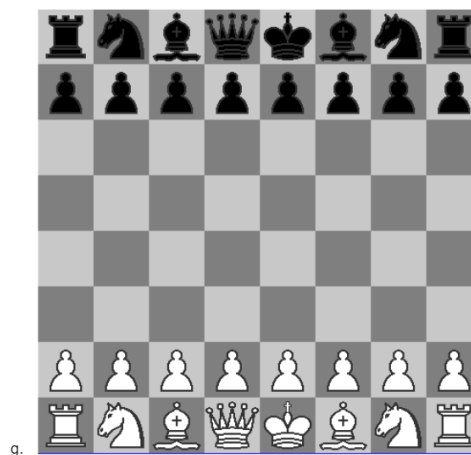


Figura 4: Ejercicios parte 2

## 4. Entregables

- Informe en Latex.
- URL: Repsitorio de GitHub.
- URL: Vídeo Explicativo(Youtube).

## 5. Equipos, materiales y temas utilizados

- Python
- Pip
- Pygame
- Virtual Environment
- GitHub
- Listas
- Ciclos
- Consicionales

## 6. URL de Repositorio Github

- Link Repositorio GitHub
- [https://github.com/Victor-Gonzalo-Maldonado-Vilca/TareaAjedrez\\_Lab05.git](https://github.com/Victor-Gonzalo-Maldonado-Vilca/TareaAjedrez_Lab05.git)

## 7. Link de Video

- Link del Vídeo de Youtube
- <https://www.youtube.com/watch?v=PpSnn4hmAWU>



Figura 5: Python

## 8. Metodología

### 8.1. Instalación de Python

#### 1. Abrir Microsoft Store:

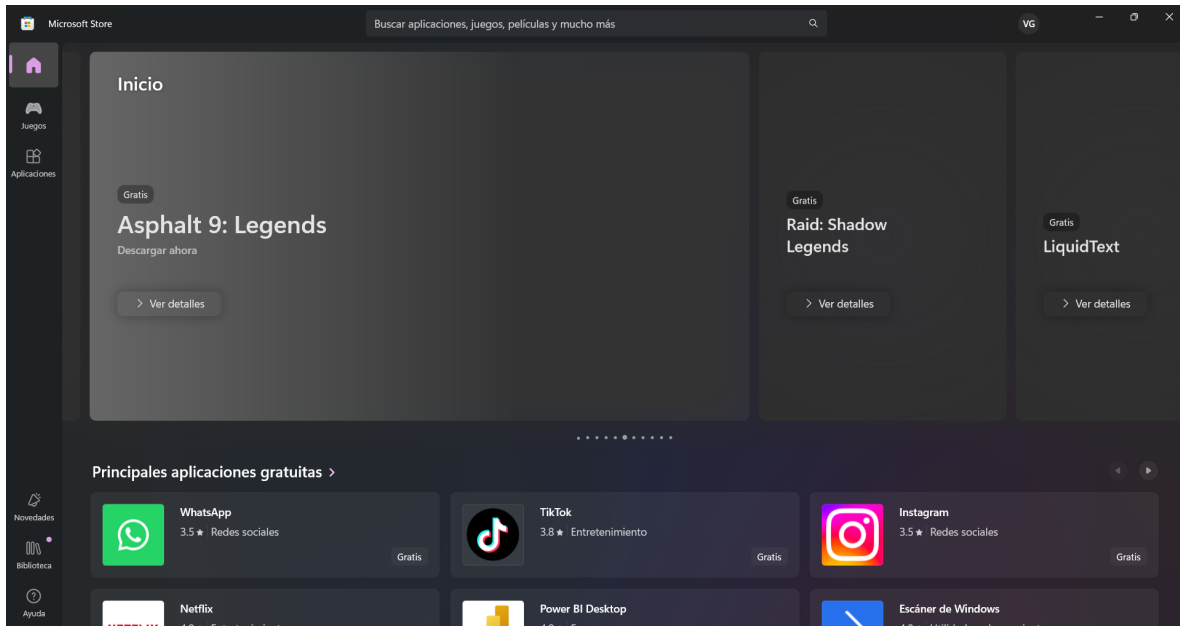


Figura 6: Microsoft Store

#### 2. Buscar Python:

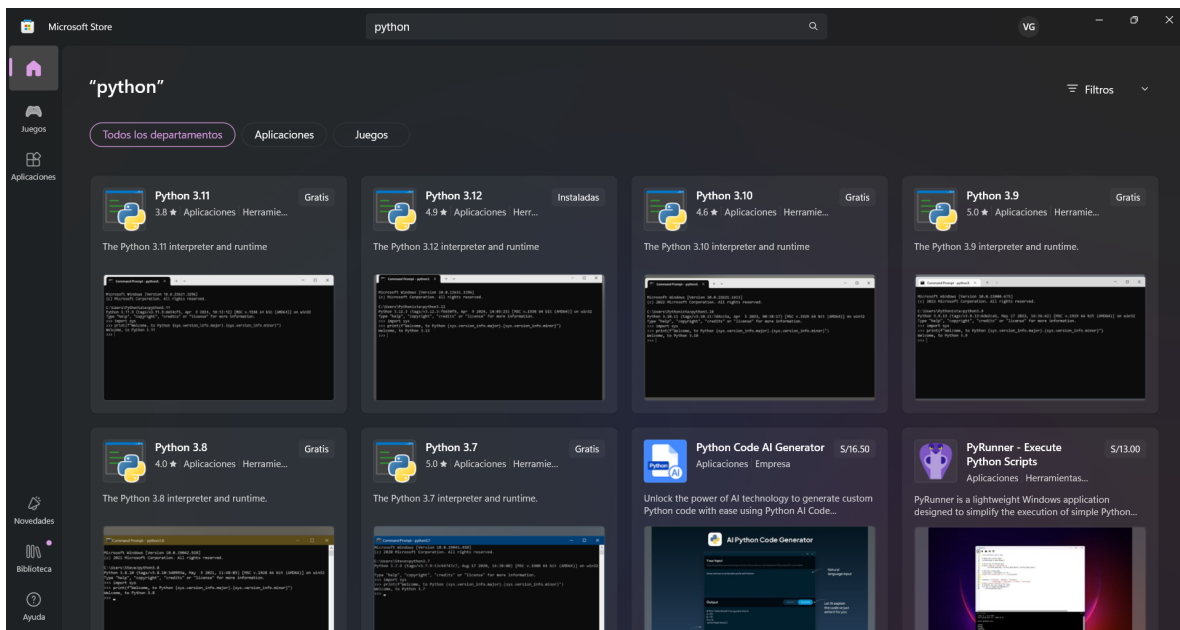


Figura 7: Microsoft Store Python

### 3. Instalar Python:

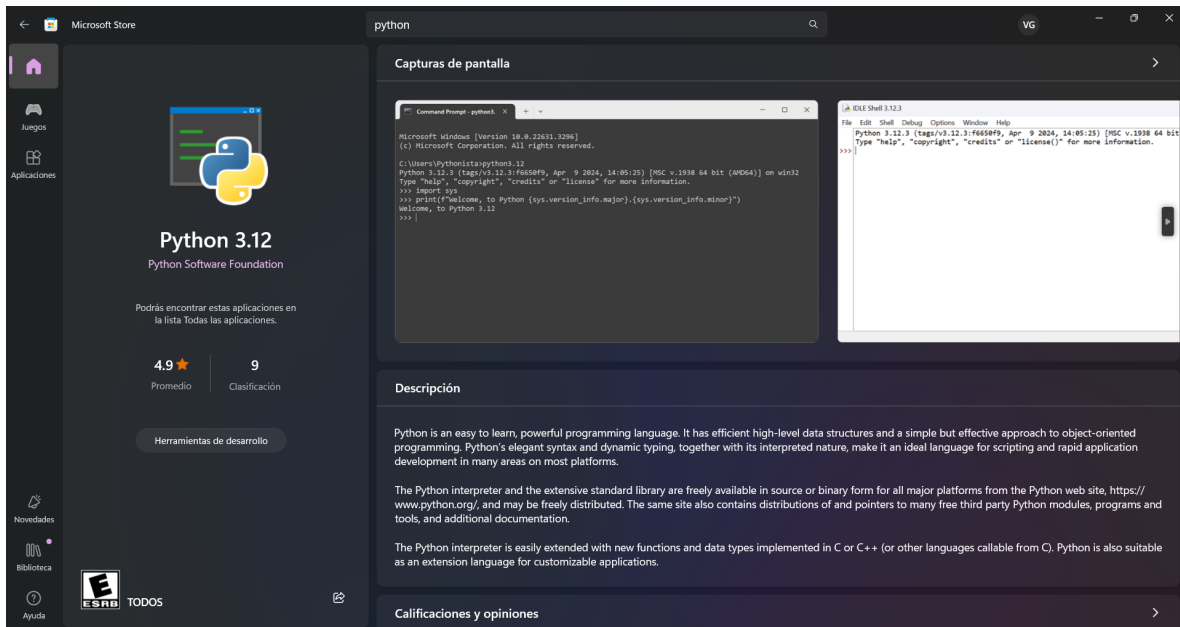


Figura 8: Instalar Python

#### 4. Verificar si se instalo correctamente: Usamos CMD para verificar.

```
python --version
```

## 8.2. Crear entorno de trabajo

### 8.2.1. Creación carpetas de Trabajo

- Carpeta lab04 (*Se usará para instalar pygame y crear entorno virtual*): usando comando:

```
mkdir lab04 && cd lab04
```

- Carpeta src (*Se usará para los archivos que se requieren en el laboratorio, de igual manera los ejercicios correspondientes*): estando en el directorio lab04 se usa el comando:

```
mkdir src && cd src
```

### 8.2.2. Uso de Pip para instalar virtualenv (*Carpeta Lab04*)

- Verificación de tener pip, comando:

```
pip --version
```

- Uso de comando pip (en mi caso ya lo tenía instalado), comando:

```
pip install virtualenv
```

- Crear entorno virtual usando virtualenv, comando:

```
virtualenv -p python3 .
```

### 8.2.3. Inicializar Git (*Carpeta src*)

- Uso del comando:

```
git init
```

### 8.3. Activar entorno Virtual (*Carpeta lab04*)

- En Linux:

```
source ../bin/activate
```

- En Windows:

```
../Scripts/activate
```

### 8.4. Obtener clases(Picture, chessPicture, etc) de Python

- Ir al Repositorio entregado por el Docente.

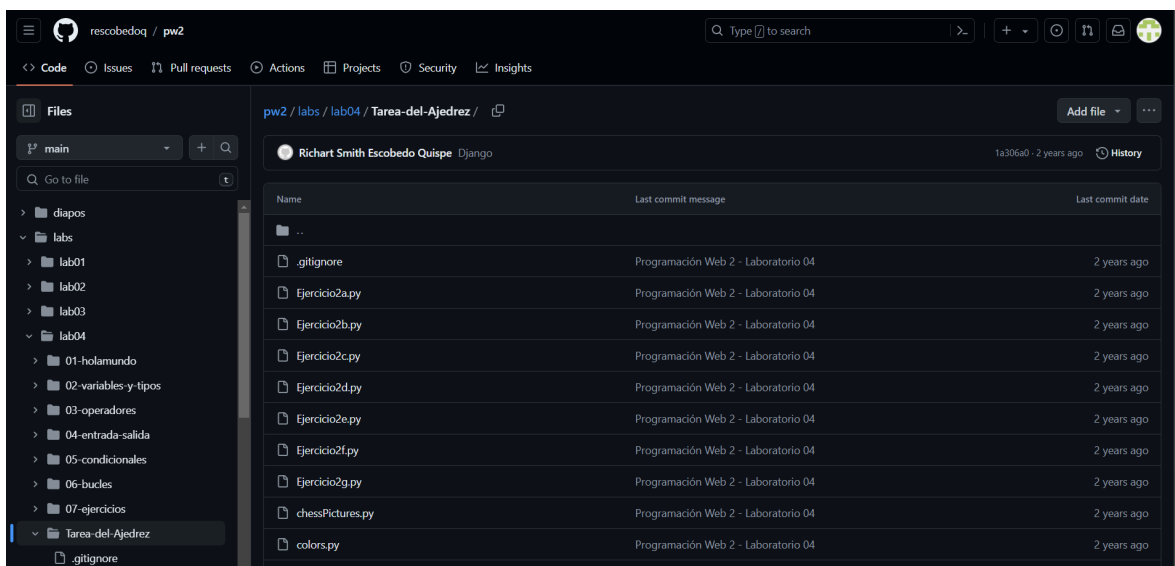


Figura 9: Repositorio Docente

- Descargar los archivos necesarios en la carpeta src.



## 8.5. Instalar Pygame

- Usar comando:

```
pip install pygame
```

## 8.6. Desactivar entorno Virtual

- Cuando ya se haya dejado de trabajar en el entorno usamos el comando:

```
deactivate
```

# 9. Desarrollo del trabajo

## 9.1. Probando funcionamiento de chessPictures y del interprete

- Prueba 1:

Listing 1: Código de prueba Python

```
from chessPictures import *  
from interpreter import draw  
x = rock  
draw(x)
```



Figura 10: prueba 1

## 9.2. Probando método Vertical Mirror

### ■ Prueba 2 (Vertical Mirror):

Listing 2: Código de prueba Python (ERROR)

```
from chessPictures import *  
from interpreter import draw  
x = verticalMirror(rock)  
draw(x)
```

### ■ Prueba 3 (Vertical Mirror):

Listing 3: Código de prueba Python (ERROR)

```
from chessPictures import *  
from interpreter import draw  
x = Picture.rock.verticalMirror()  
draw(x)
```

### ■ Prueba 4 *Vertical Mirror*: NORMAL:

Listing 4: Código de prueba Python

```
from chessPictures import *  
from interpreter import draw  
x = knight  
draw(x)
```



Figura 11: prueba 4

## VERTICAL MIRROR:

Listing 5: Código de prueba Python

```
from chessPictures import *
from interpreter import draw
x = rock.verticalMirror()
draw(x)
```

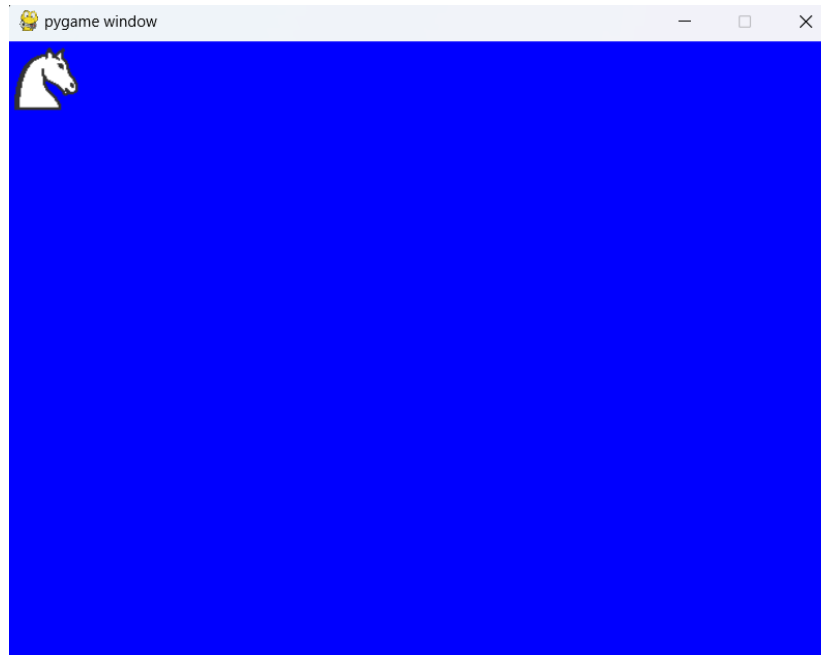


Figura 12: prueba 4

## 9.3. Implementación de los Métodos en la clase Picture:

### 9.3.1. horizontalMirror()

- **Descripción:** El método horizontalMirror devuelve la imagen reflejada horizontalmente de una imagen representada por la lista self.img. Itera sobre los elementos de self.img, los agrega en orden inverso a una nueva lista llamada horizontal, y retorna esta lista como una imagen reflejada horizontalmente.
- **Código - sin definición de Listas por Comprensión:**

Listing 6: Método horizontalMirror()

```
def horizontalMirror(self):
    """ Devuelve el espejo horizontal de la imagen """
    horizontal = []
    longitud = len(self.img)
    for i in range (longitud):
        horizontal.append(self.img[longitud - i - 1])
    return Picture(horizontal)
```

- **Código - con definición de Listas por Comprensión:**

Listing 7: Método horizontalMirror()

```
def horizontalMirror(self):
    """ Devuelve el espejo horizontal de la imagen """
    longitud = len(self.img)
    horizontal = [self.img[longitud - i - 1] for i in range (longitud)]
    return Picture(horizontal)
```

- **Ejecución:**

Listing 8: Prueba de horizontalMirror()

```
from chessPictures import *
from interpreter import draw
x = rock.horizontalMirror()
draw(x)
```

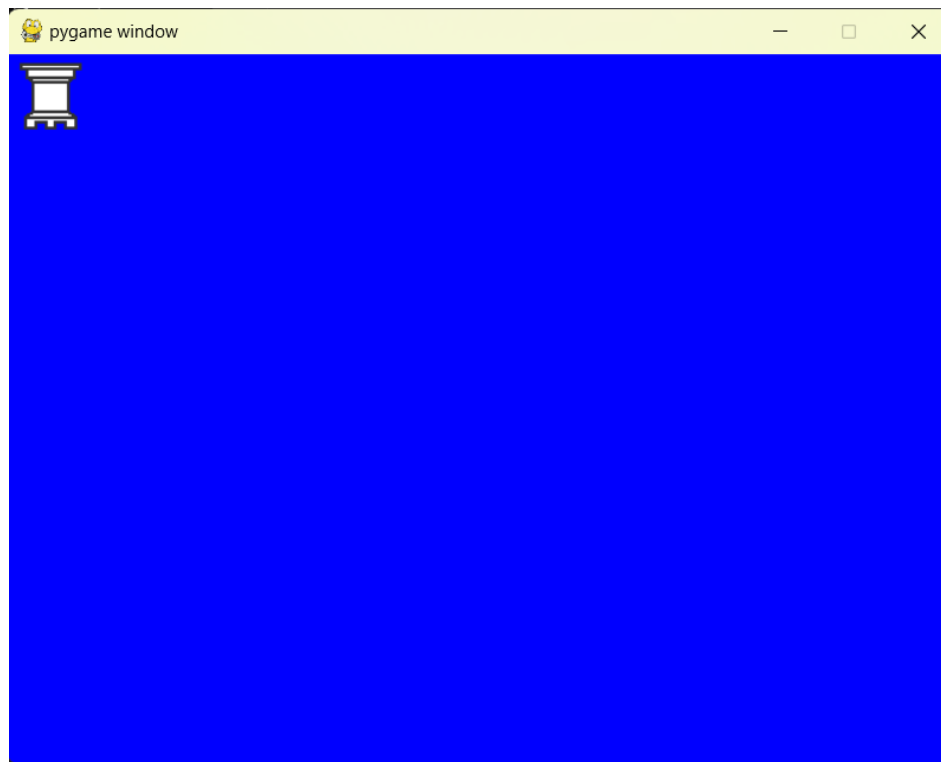


Figura 13: Método horizontalMirror()

### 9.3.2. negative()

- **Descripción:** Este método negative retorna el negativo de una imagen representada por la lista self.img. Recorre cada fila de la imagen, luego cada píxel en esa fila, invierte su color utilizando la función invColor, y agrega el píxel invertido a una nueva fila en la lista negativo.

Finalmente, devuelve una nueva imagen representada por la lista negativo, que es el negativo de la imagen original. En resumen, este método genera el negativo de una imagen al invertir los colores de todos los píxeles en cada fila de la imagen.

■ **Código - sin definición de Listas por Comprensión:**

Listing 9: Método negative()

```
def negative(self):
    """ Devuelve un negativo de la imagen """
    negativo = []
    for value in self.img:
        valueNegative = ""
        for i in range(len(value)):
            valueNegative += self._invColor(value[i])
        negativo.append(valueNegative)
    return Picture(negativo)
```

■ **Código - con definición de Listas por Comprensión:**

Listing 10: Método negative()

```
def negative(self):
    """ Devuelve un negativo de la imagen """
    negativo = [''.join(self._invColor(pixel) for pixel in value) for value in
                self.img]
    return Picture(negativo)
```

■ **Ejecución:**

Listing 11: Prueba de negative()

```
from chessPictures import *
from interpreter import draw
x = king.negative()
draw(x)
```

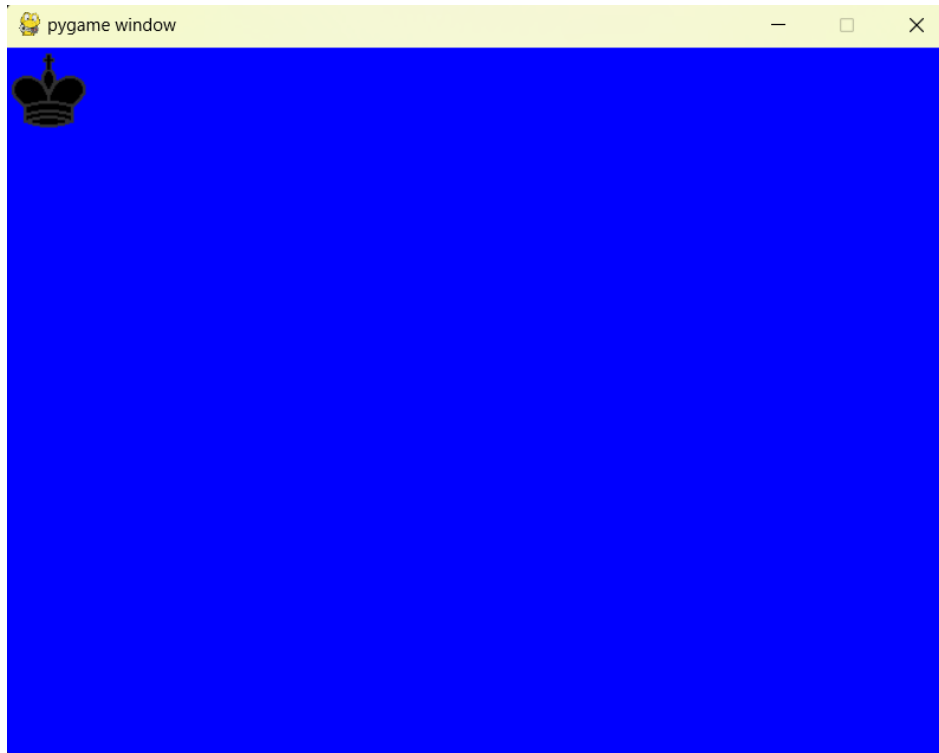


Figura 14: Método negative()

### 9.3.3. join()

- **Descripción:** El método join devuelve una nueva figura que combina la figura actual con otra figura proporcionada como argumento, colocando la figura del argumento al lado derecho de la figura actual. Utiliza la función zip para combinar las filas de ambas figuras y luego une los elementos correspondientes de cada fila mediante la concatenación. Finalmente, crea una nueva imagen con las filas combinadas y la devuelve.
- **Código - sin definición de Listas por Comprensión:**

Listing 12: Método join()

```
def join(self, p):
    """ Devuelve una nueva figura poniendo la figura del argumento
        al lado derecho de la figura actual """
    alado = []
    i = 0
    for value in self.img:
        alado.append(value)
    for value in p.img:
        alado[i] += value
        i += 1
    return Picture(alado)
```

- **Código - con definición de Listas por Comprensión:**

Listing 13: Método join()

```
def join(self, p):
    """ Devuelve una nueva figura poniendo la figura del argumento
        al lado derecho de la figura actual """
    alado = [value1 + value2 for value1,value2 in zip(self.img,p.img)]
    return Picture(alado)
```

- **Ejecución:**

Listing 14: Prueba de join()

```
from chessPictures import *
from interpreter import draw
x = queen.join(square)
draw(x)
```

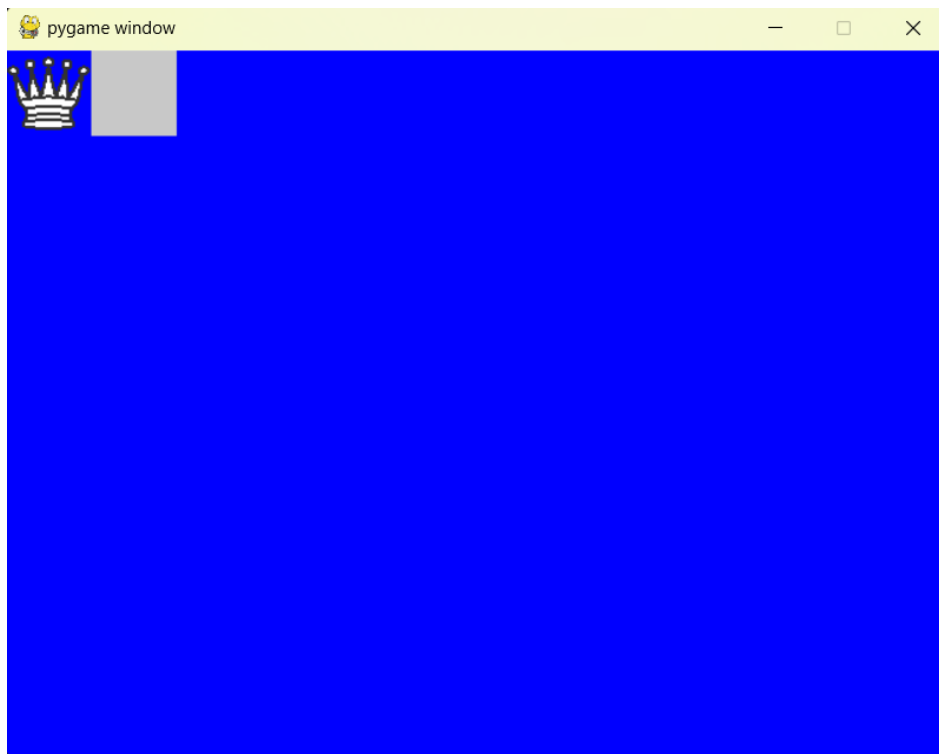


Figura 15: Método join()

#### 9.3.4. up()

- **Descripción:** La función up de la clase Picture toma otra instancia de Picture como parámetro, representada por p. Internamente, combina las imágenes de la instancia actual y la instancia p, colocando la imagen de p encima de la imagen de la instancia actual. Luego, devuelve un nuevo objeto Picture con esta combinación de imágenes.

■ **Código:**

Listing 15: Método up()

```
def up(self, p):
    encima = p.img + self.img
    return Picture(encima)
```

■ **Ejecución:**

Listing 16: Prueba up()

```
from chessPictures import *
from interpreter import draw
x = queen.up(rock)
draw(x)
```

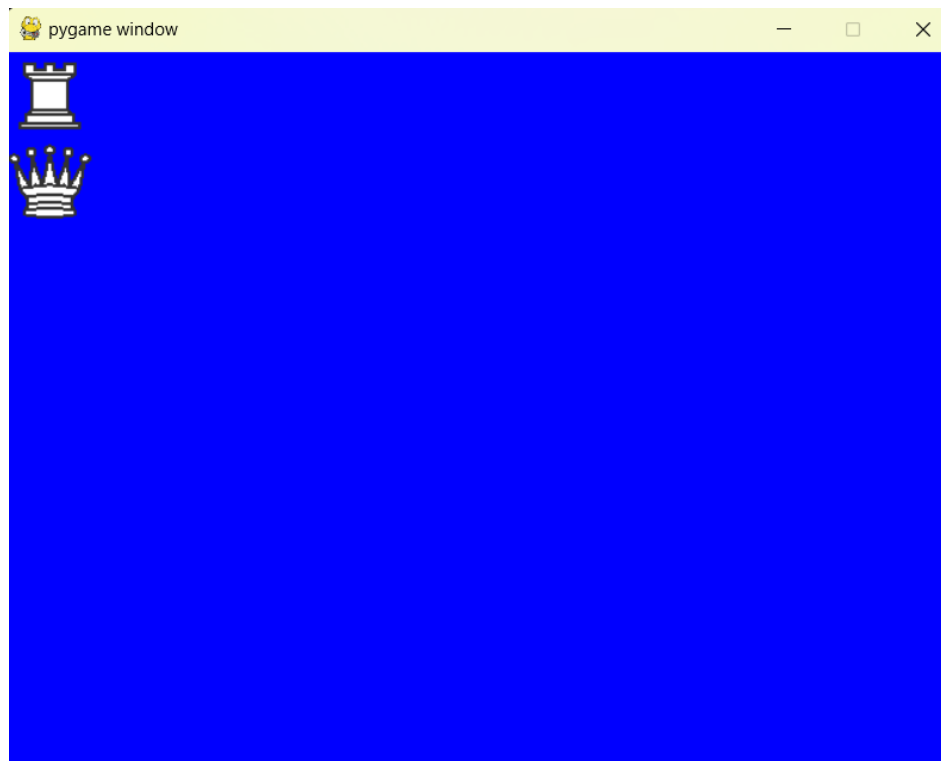


Figura 16: Método up()

### 9.3.5. under()

- **Descripción:** El método under de la clase Picture crea una nueva imagen colocando la imagen de otro objeto Picture sobre la imagen de la instancia actual. Si las imágenes no tienen la misma longitud, simplemente devuelve la imagen del objeto Picture pasado como argumento. En caso contrario, combina las imágenes superponiendo los píxeles de ambas imágenes y retorna un nuevo objeto Picture con esta composición.



- Código - sin definición de Listas por Comprensión:

Listing 17: Método under()

```
def under(self, p):  
    """ Devuelve una nueva figura poniendo la figura p sobre la  
        figura actual """  
    sobre = []  
    if(len(p.img) != len(self.img)):  
        return Picture(p.img)  
    for value1,value2 in zip(self.img,p.img):  
        cadena = ""  
        for i in range(len(value1)):  
            if(value2[i] == ' '):  
                cadena += value1[i]  
            else:  
                cadena += value2[i]  
        sobre.append(cadena)  
    return Picture(sobre)
```

- Código - con definición de Listas por Comprensión:

Listing 18: Método under()

```
def under(self, p):  
    """ Devuelve una nueva figura poniendo la figura p sobre la  
        figura actual """  
    sobre = [''.join(value1[i] if value2[i] == ' ' else value2[i] for i in  
        range(len(value1))) for value1,value2 in zip(self.img,p.img)]  
    return Picture(sobre) if len(self.img) == len(p.img) else Picture(p.img)
```

- Ejecución:

Listing 19: Prueba under()

```
from chessPictures import *  
from interpreter import draw  
x = square.under(rock)  
draw(x)
```

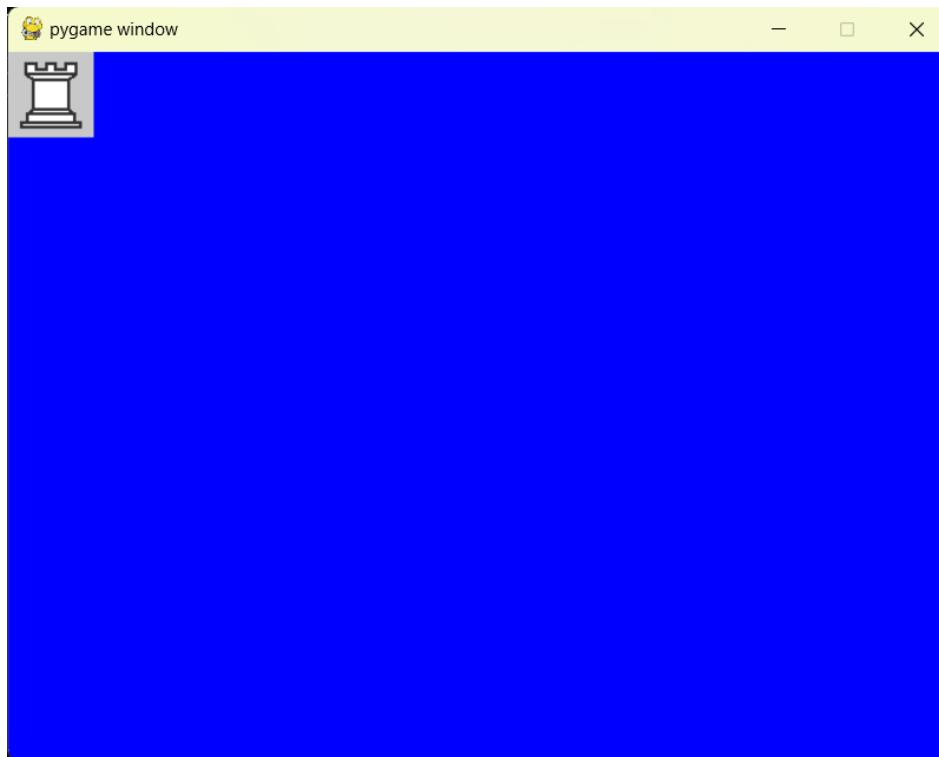


Figura 17: Método under()

### 9.3.6. horizontalRepeat()

- **Descripción:** horizontalRepeat es un método de la clase Picture que devuelve una nueva figura que repite horizontalmente la figura actual un número específico de veces. Para lograr esto, el método toma el número de repeticiones como argumento y construye una nueva figura donde cada fila de la figura original se concatena consigo misma la cantidad de veces especificada. Esto permite crear una nueva figura que tenga la misma forma que la figura original, pero que se extiende horizontalmente según el número de repeticiones especificado.
- **Código - sin definición de Listas por Comprensión:**

Listing 20: Método horizontalRepeat()

```
def horizontalRepeat(self, n):
    """ Devuelve una nueva figura repitiendo la figura actual al costado
        la cantidad de veces que indique el valor de n """
    repeath = []
    for value in self.img:
        cadena = ""
        for i in range(n):
            cadena += value
        repeath.append(cadena)
    return Picture(repeath)
```

- **Código - con definición de Listas por Comprensión:**

Listing 21: Método horizontalRepeat()

```
def horizontalRepeat(self, n):
    """ Devuelve una nueva figura repitiendo la figura actual al costado
        la cantidad de veces que indique el valor de n """
    repeath = [''].join(value for i in range(n)) for value in self.img]
    return Picture(repeath)
```

- **Ejecución:**

Listing 22: Prueba horizontalRepeat()

```
from chessPictures import *
from interpreter import draw
x = knight.horizontalRepeat(7)
draw(x)
```

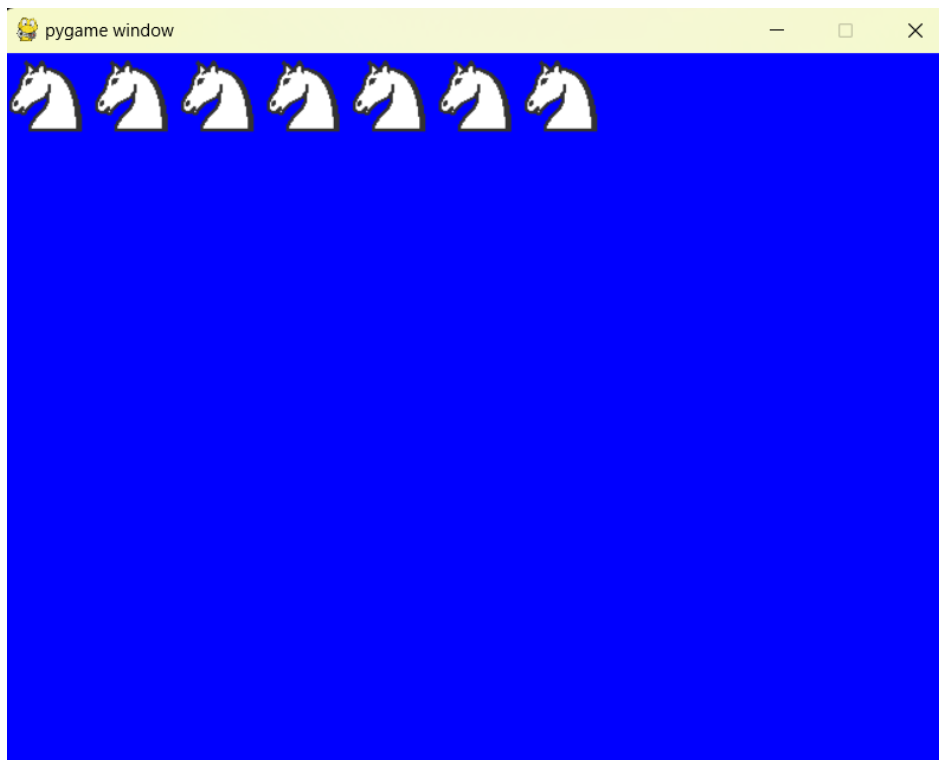


Figura 18: Método horizontalRepeat()

### 9.3.7. verticalRepeat()

- **Descripción:** verticalRepeat genera una nueva figura replicando verticalmente la figura actual según un número específico de repeticiones. Esto significa que la imagen se apila una encima de la otra, creando una imagen más alta.

- Código - sin definición de Listas por Comprensión:

Listing 23: Método verticalRepeat()

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

- Código - con definición de Listas por Comprensión:

Listing 24: Método verticalRepeat()

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

- Ejecución:

Listing 25: Prueba verticalRepeat()

```
from chessPictures import *  
from interpreter import draw  
x = king.verticalRepeat(5)  
draw(x)
```

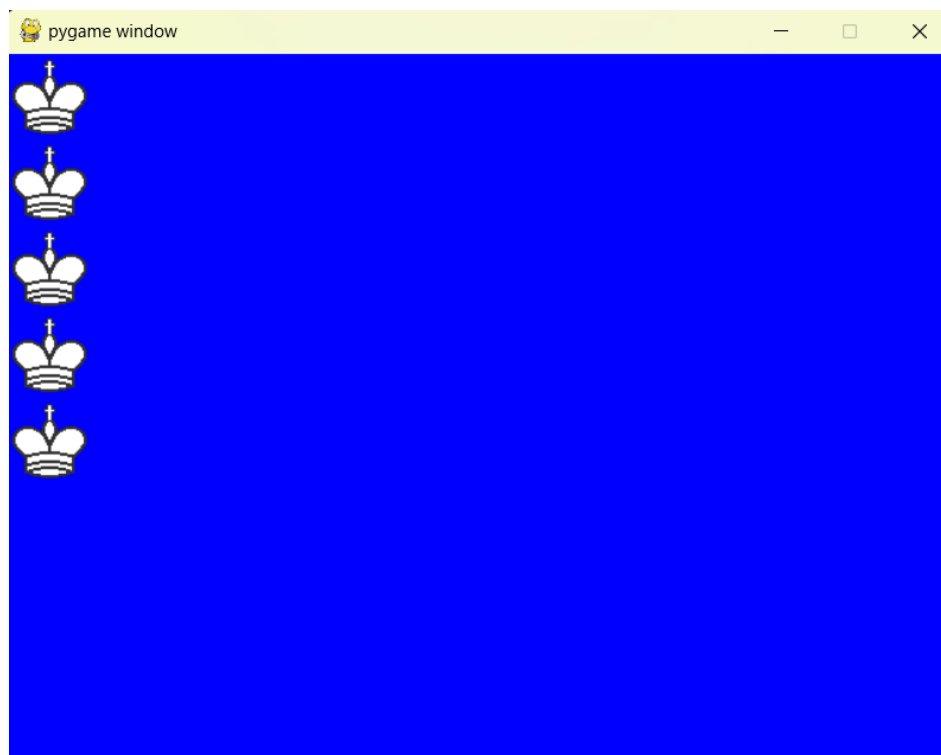


Figura 19: Método verticalRepeat()

### 9.3.8. rotate()

- **Descripción:** rotate es un método que devuelve una nueva figura rotada en 90 grados. Esto significa que los elementos de la figura se reorganizan de manera que cada fila se convierte en una columna en la nueva figura, y las filas originales se reorganizan en orden inverso.
- **Código - sin definición de Listas por Comprensión:**

Listing 26: Método rotate()

```
def rotate(self):
    """Devuelve una figura rotada en 90 grados, puede ser en sentido horario
    o antihorario"""
    rotar = []
    for i in range(len(self.img[0])):
        cadena = ""
        for c in range(len(self.img) - 1, -1, -1):
            value = self.img[c]
            cadena += value[i]
        rotar.append(cadena)
    return Picture(rotar)
```

- **Código - con definición de Listas por Comprensión:**

Listing 27: Método rotate()

```
def rotate(self):
    """Devuelve una figura rotada en 90 grados, puede ser en sentido horario
    o antihorario"""
    rotar = [''.join(self.img[c][i] for c in range(len(self.img) - 1, -1, -1)) for i
              in range(len(self.img[0]))]
    return Picture(rotar)
```

- **Ejecución:**

Listing 28: Prueba rotate()

```
from chessPictures import *
from interpreter import draw
x = queen.rotate()
draw(x)
```

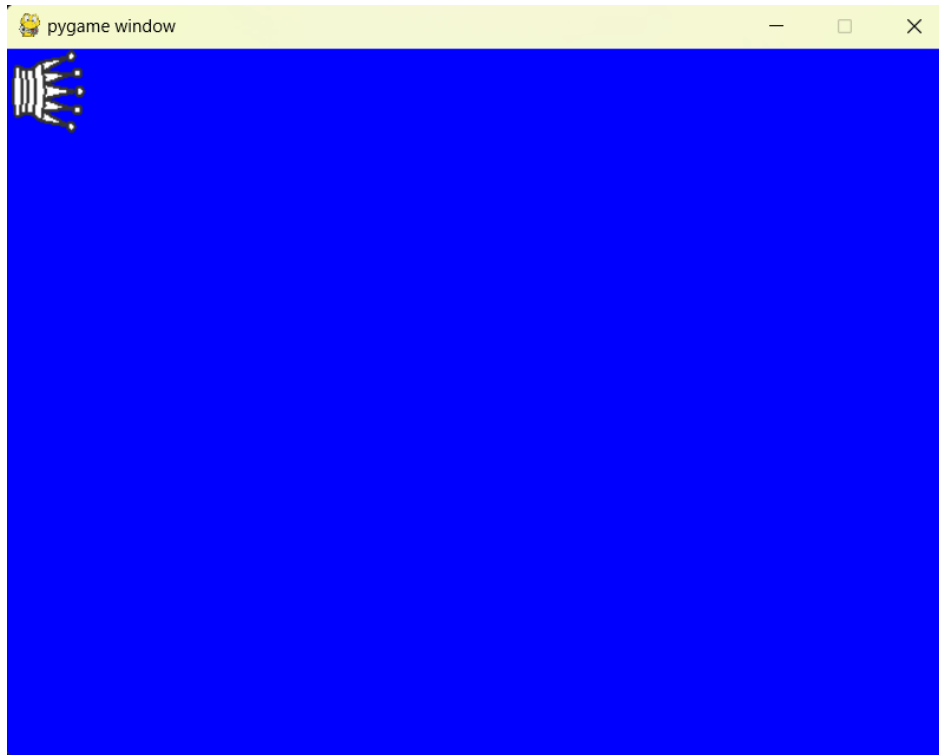


Figura 20: Método rotate()

### 9.3.9. EXTRA: verticalMirror

*El Método ya fue implementado por el docente, en esta sección se dará a conocer otra manera de la implementación del Método*

- Código - con definición de Listas por Comprensión:

Listing 29: Método verticalMirror()

```
def verticalMirror(self):
    """ Devuelve el espejo vertical de la imagen """
    vertical = [value[::-1] for value in self.img]
    return Picture(vertical)
```

- Ejecución:

Listing 30: Prueba verticalMirror()

```
from chessPictures import *
from interpreter import draw
x = knight.join(rock).verticalMirror()
draw(x)
```

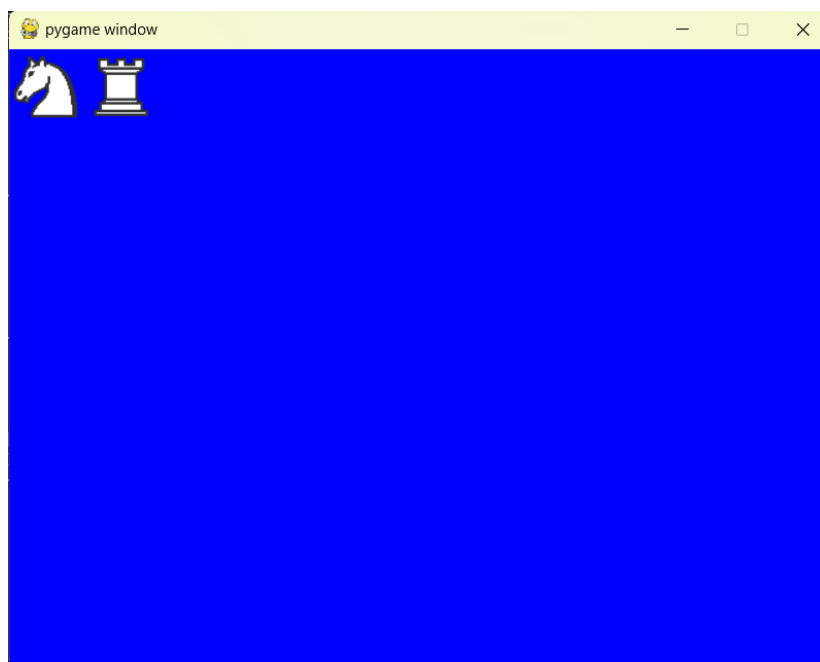


Figura 21: Gráfica Normal

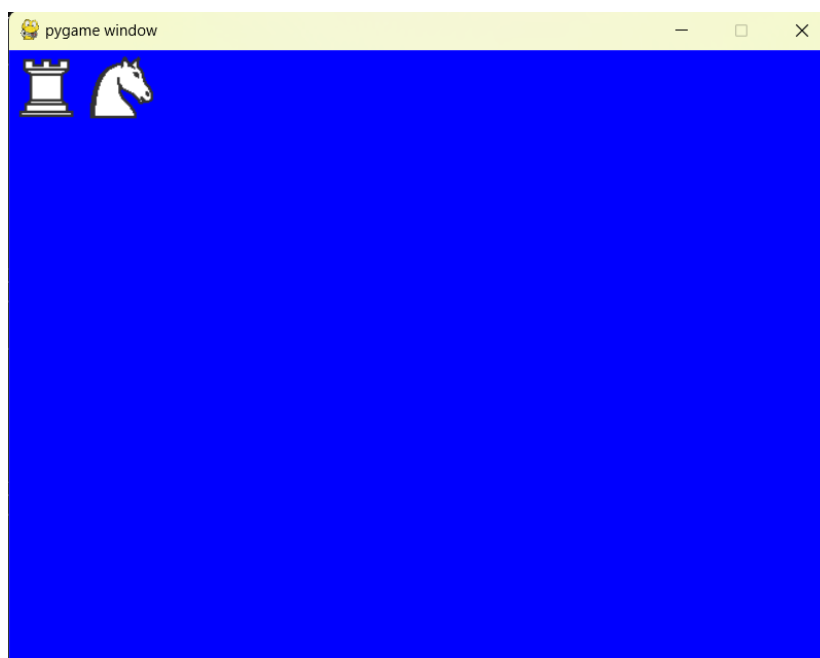


Figura 22: Método verticalMirror()

## 9.4. EJERCICIOS

*Lograr Gráficar las imágenes que se muestran en el documento del Docente*

### 9.4.1. Ejercicio 1 (A)

- Código:

Listing 31: Ejercicio2a

```
from interpreter import draw
from chessPictures import *
fila = knight.join(knight.negative())
draw(fila.negative().up(fila))
```

- Ejecución:

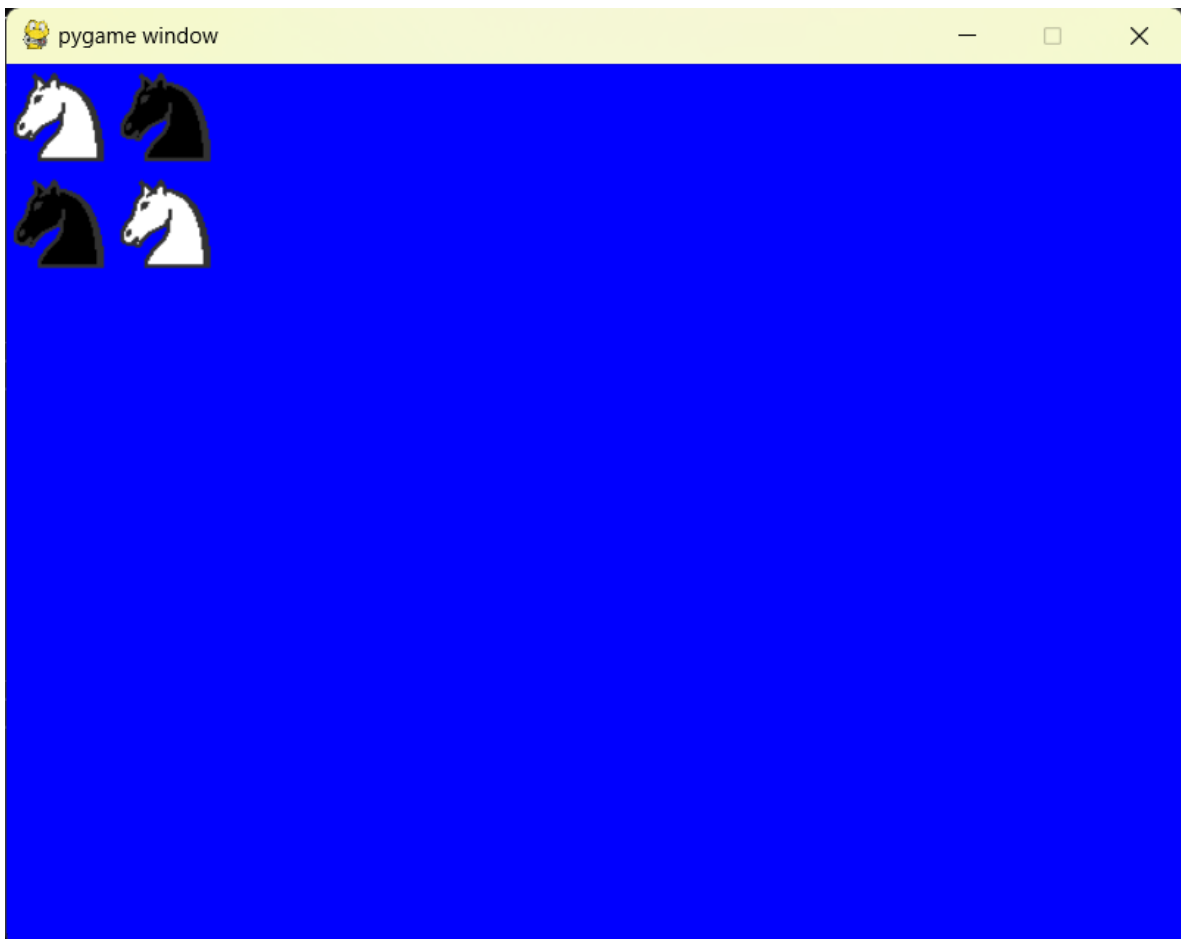


Figura 23: Ejercicio 1



#### 9.4.2. Ejercicio 2 (B)

- Código:

Listing 32: Ejercicio2b

```
from interpreter import draw
from chessPictures import *
fila = knight.join(knight.negative())
draw(fila.verticalMirror().up(fila))
```

- Ejecución:

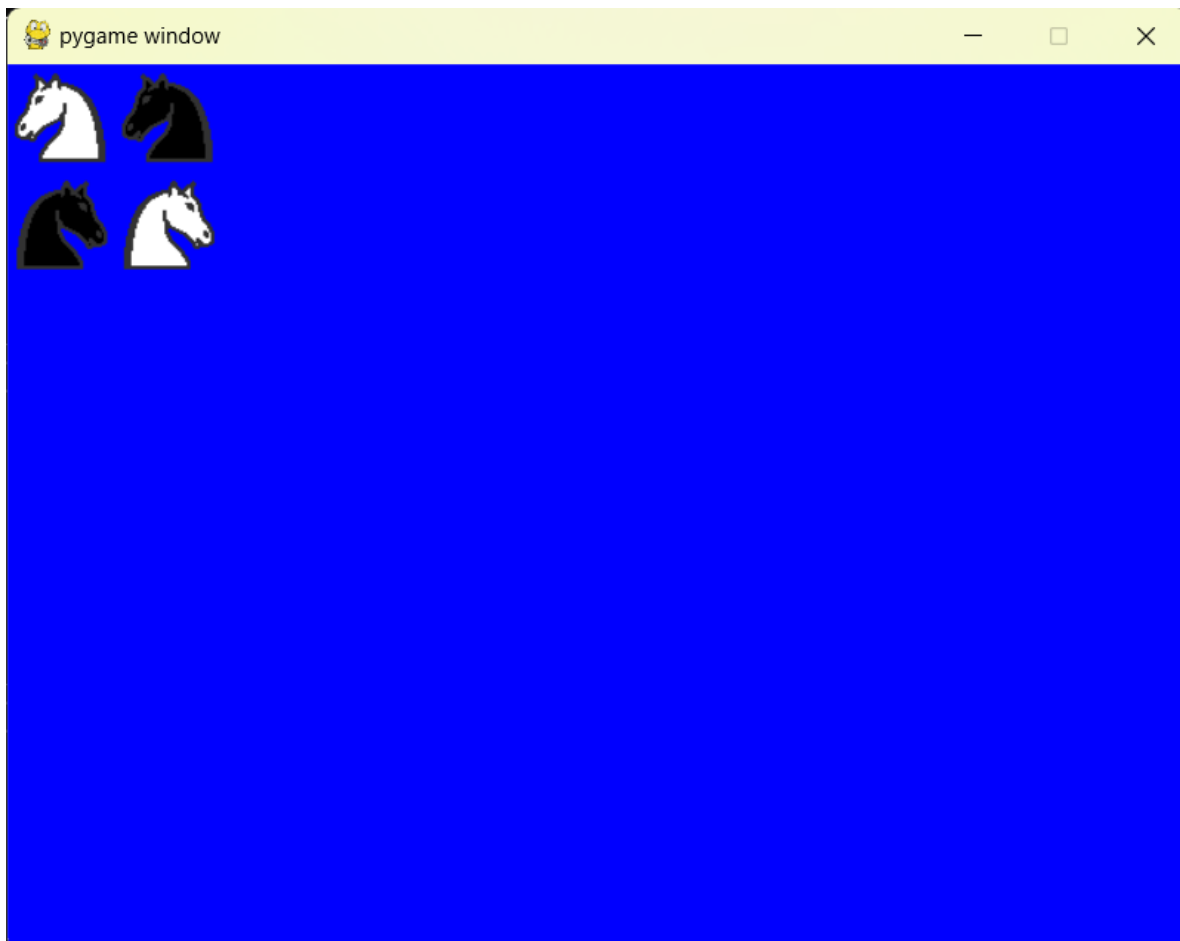


Figura 24: Ejercicio 2

#### 9.4.3. Ejercicio 3 (C)

- Código:

Listing 33: Ejercicio2c

```
from interpreter import draw
from chessPictures import *
fila = queen.horizontalRepeat(4)
draw(fila)
```

- Ejecución:



Figura 25: Ejercicio 3

#### 9.4.4. Ejercicio 4 (D)

- Código:

Listing 34: Ejercicio2d

```
from interpreter import draw
from chessPictures import *
fila = square.join(square.negative()).horizontalRepeat(4)
draw(fila)
```

- Ejecución:

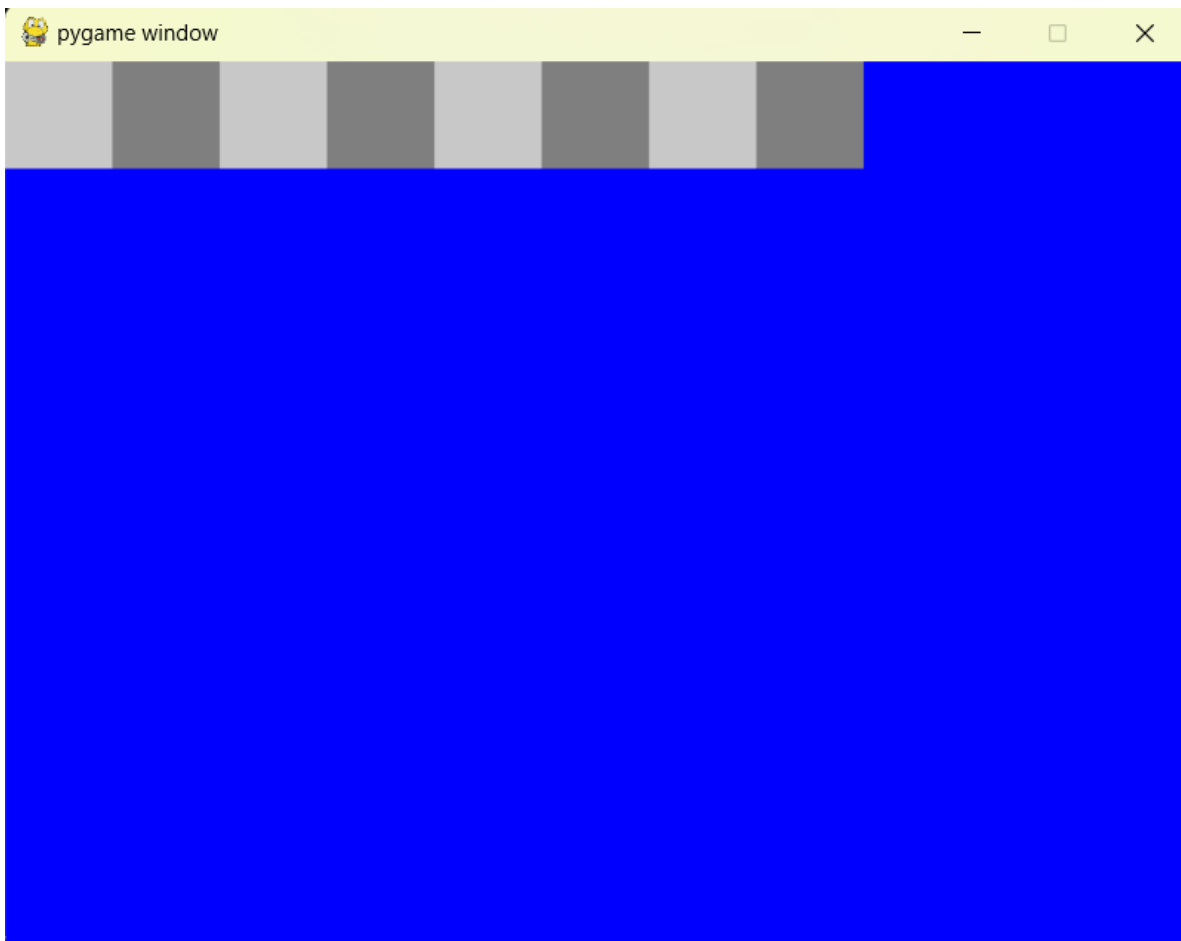


Figura 26: Ejercicio 4

#### 9.4.5. Ejercicio 5 (E)

- Código:

Listing 35: Ejercicio2e

```
from interpreter import draw
from chessPictures import *
fila = square.join(square.negative()).negative().horizontalRepeat(4)
draw(fila)
```

- Ejecución:

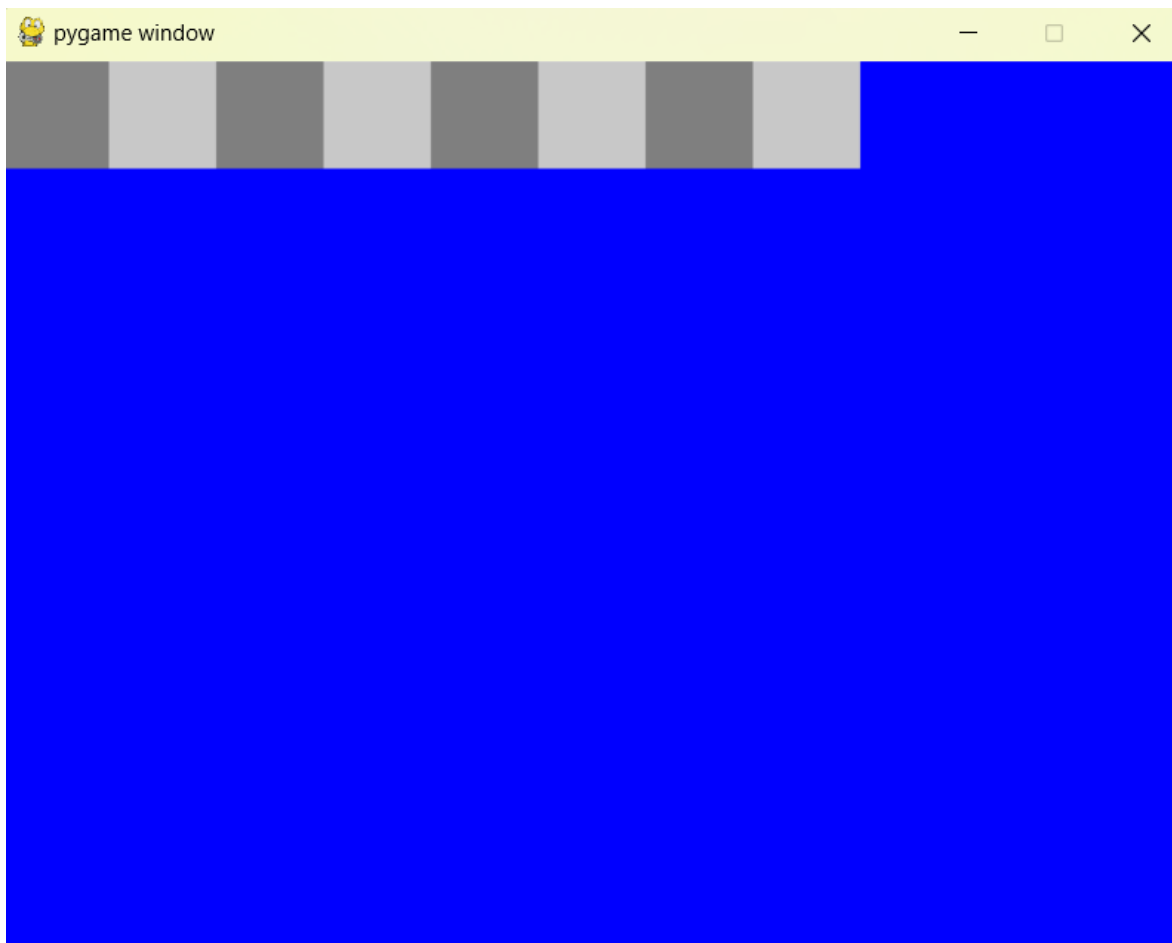


Figura 27: Ejercicio 5

#### 9.4.6. Ejercicio 6 (F)

- Código:

Listing 36: Ejercicio2f

```
from interpreter import draw
from chessPictures import *
fila = square.join(square.negative()).horizontalRepeat(4)
draw(fila.negative().up(fila).verticalRepeat(2))
```

- Ejecución:

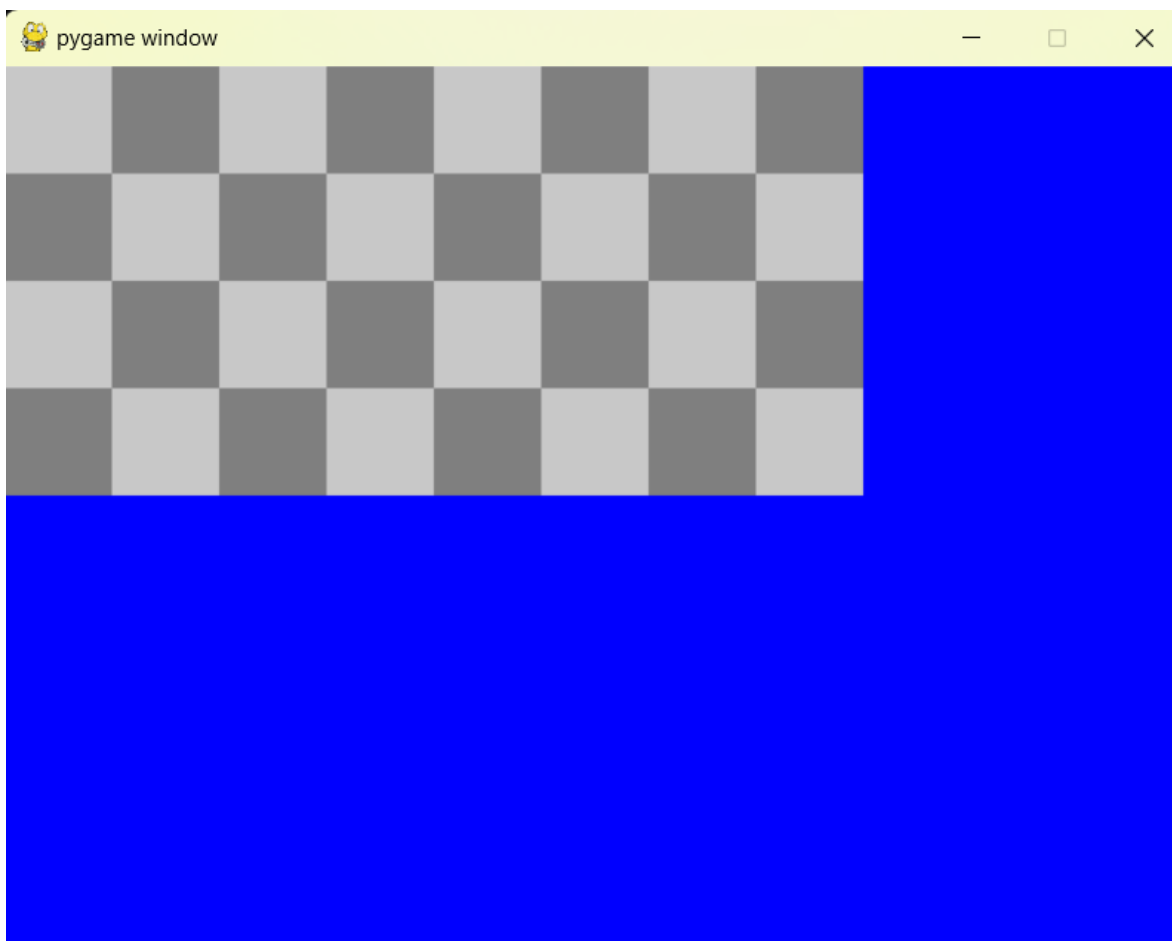


Figura 28: Ejercicio 6

#### 9.4.7. Ejercicio 7 (G)

■ Código:

Listing 37: Ejercicio2g

```
from interpreter import draw
from chessPictures import *
fila = square.join(square.negative()).horizontalRepeat(4)
fichas =
    rock.join(knight).join(bishop).join(queen).join(king).join(bishop).join(knight).join(rock)
peones = pawn.horizontalRepeat(8)
fichasBlancas = fila.negative().under(fichas).up(fila.under(peones))
fichasNegras =
    fila.negative().under(peones.negative()).up(fila.under(fichas.negative()))
tablero =
    fichasBlancas.up(fila.negative().up(fila).verticalRepeat(2)).up(fichasNegras)
draw(tablero)
```

■ Ejecución:

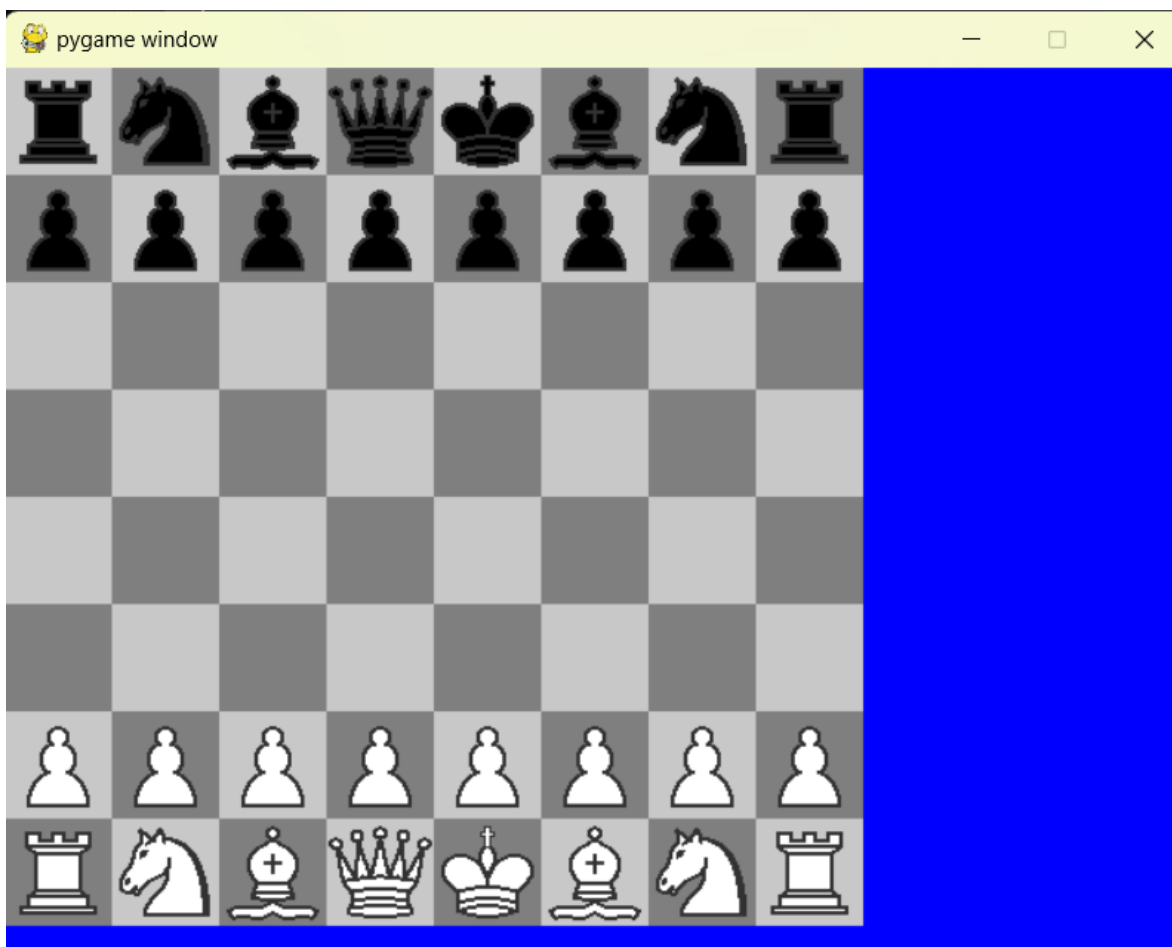


Figura 29: Ejercicio 7

## 9.5. Uso de GitHub

### 9.5.1. Usuario de GitHub

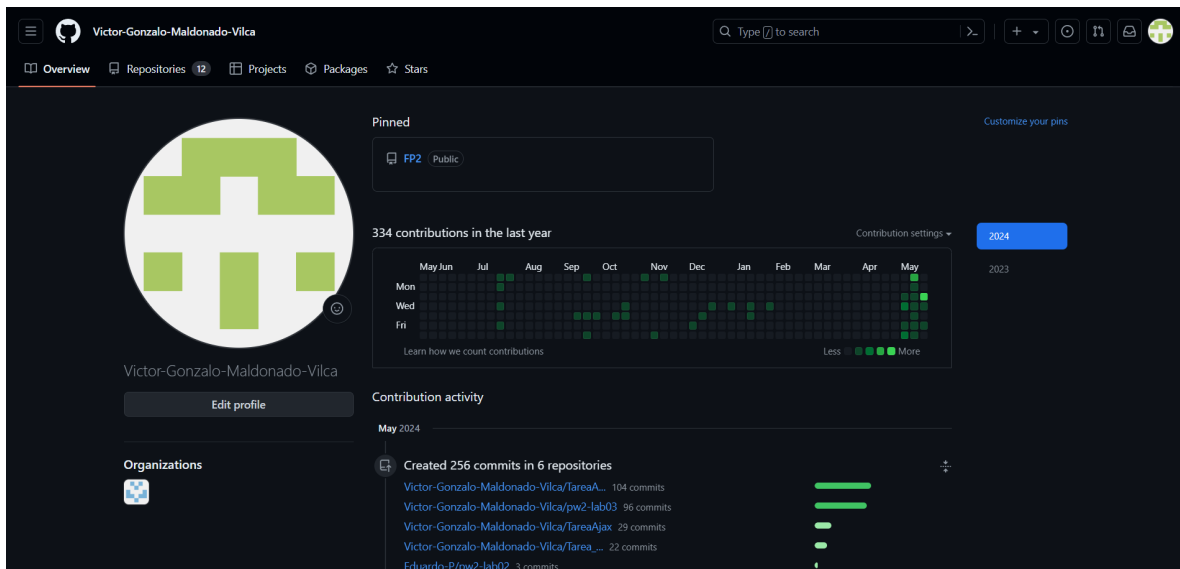


Figura 30: Usuario

### 9.5.2. Creación de un Nuevo Repositorio

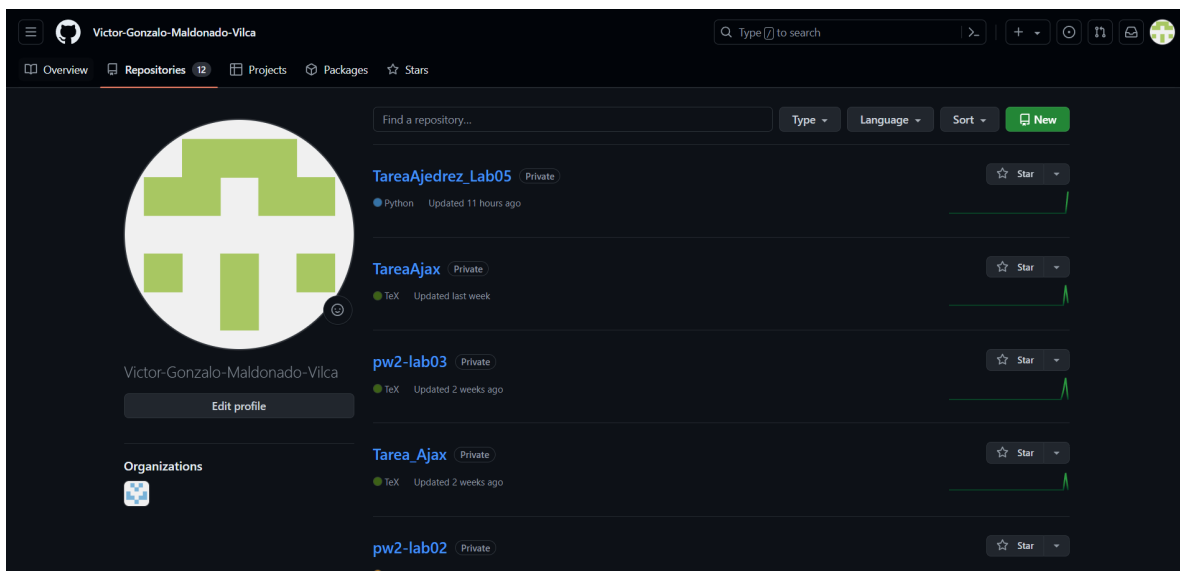


Figura 31: Crear Repositorio TareaAjedrez\_lab05

### 9.5.3. Comandos de Configuración

Listing 38: Configuración Inicial

```
echo "# TareaAjedrez_lab05" >> README.md
git init
git add README.md
git commit -m "first commit"
git branch -M master
git remote add origin
    https://github.com/Victor-Gonzalo-Maldonado-Vilca/TareaAjedrez_Lab05.git
git push -u origin master
```

### 9.5.4. Implementación de Readme.md

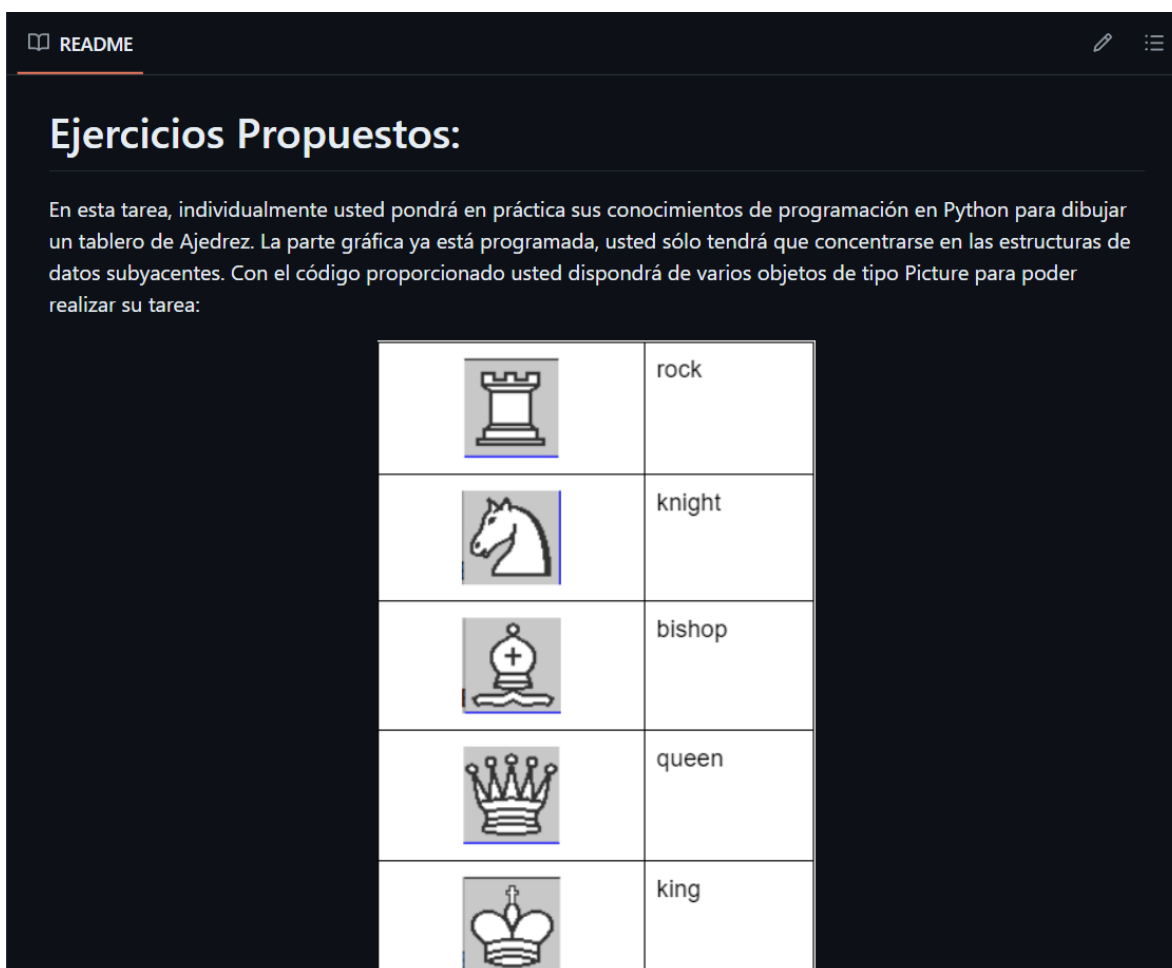


Figura 32: README.md



### 9.5.5. Registro de cambios en mi código

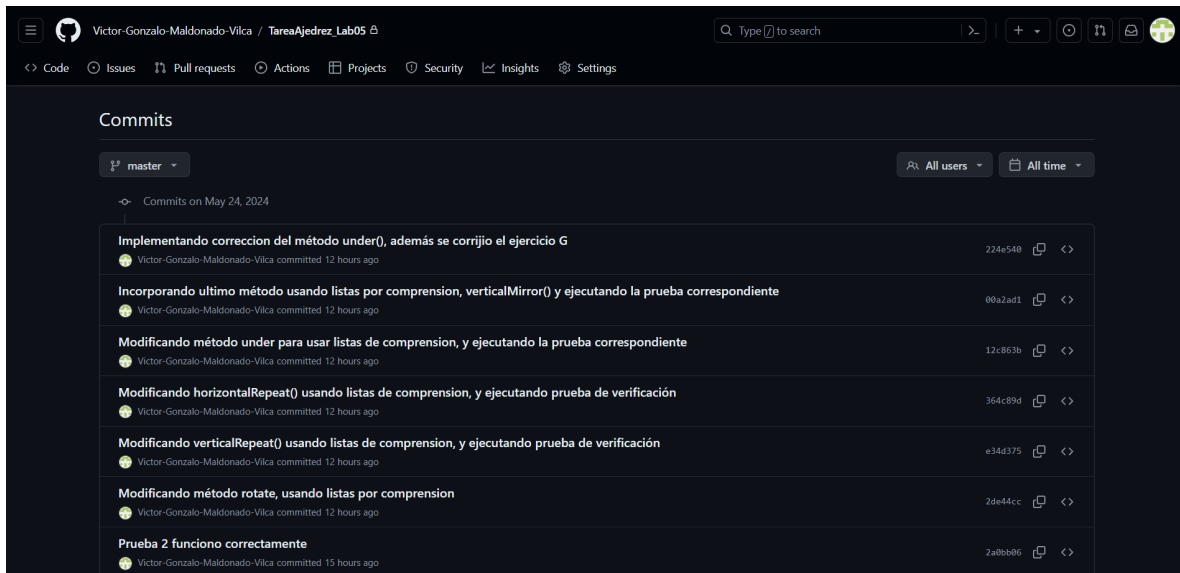


Figura 33: Commits

### 9.5.6. Repositorio

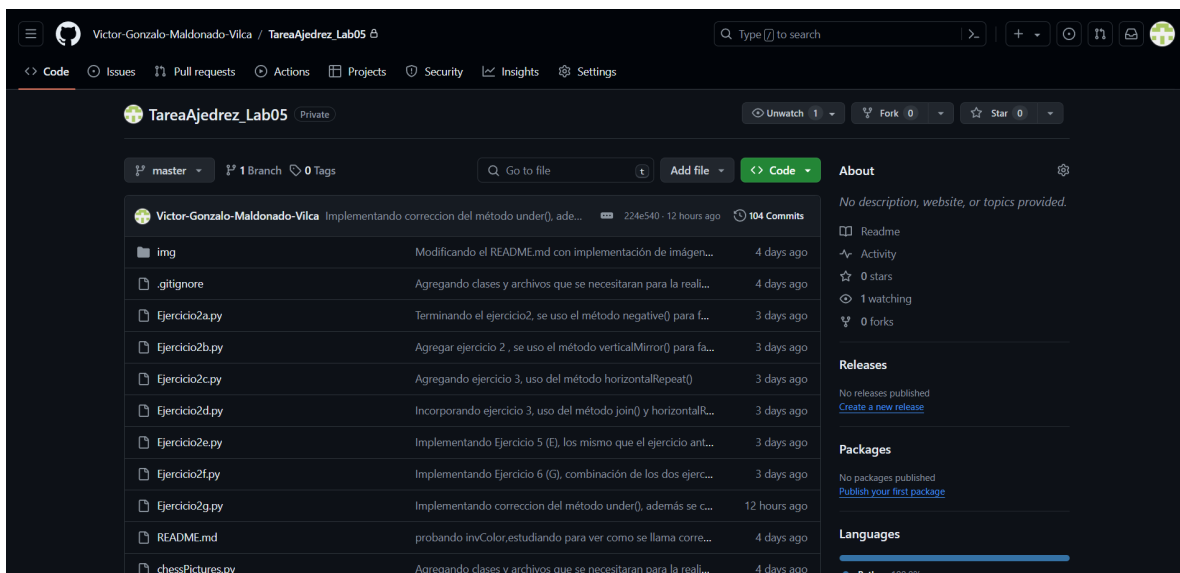


Figura 34: Repositorio AjedrezTarea\_lab05

### 9.5.7. Proyecto compartido con el Docente de github

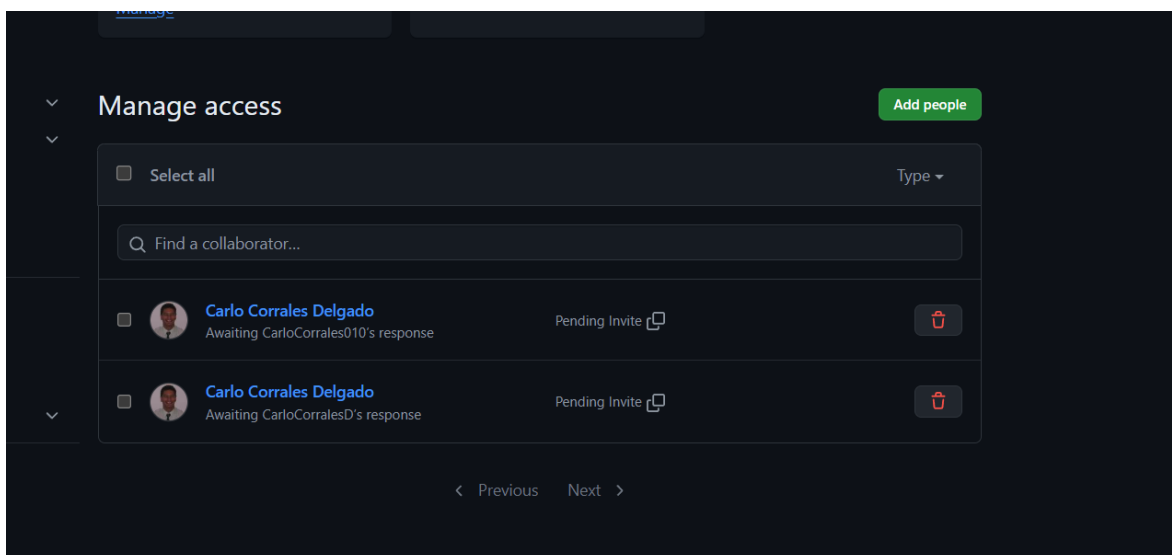


Figura 35: Compartir con el Docente

## 10. Recomendaciones

- Utilizar nombres de variables y funciones descriptivos para mejorar la legibilidad y comprensión del código.
- Aprovechar las características del lenguaje Python, como la comprensión de listas, generadores y decoradores, para escribir código más eficiente y elegante.
- Utilizar entornos virtuales para aislar las dependencias de tus proyectos y evitar conflictos entre diferentes versiones de paquetes.
- Estudiar y experimentar para comprender mejor cómo trabajar con la clase Picture.

## 11. Conclusiones

- La creación de entornos virtuales utilizando herramientas como virtualenv es fundamental para mantener un ambiente de desarrollo limpio y aislado para cada proyecto.
- Pygame es una excelente opción para el desarrollo de juegos en Python, ofreciendo capacidades para crear interfaces gráficas, manejar eventos de entrada y renderizar gráficos de manera eficiente.
- Picture es una biblioteca útil para la representación gráfica de figuras de ajedrez en Python, ofreciendo funciones para cargar, procesar y renderizar estas figuras de manera eficiente.

### 11.1. Rúbrica para el contenido del Informe y demostración

- El alumno debe marcar o dejar en blanco en celdas de la columna **Checklist** si cumple 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 los ítems.
- 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
<b>1. GitHub</b>	Hay enlace URL activo del directorio para el laboratorio hacia su repositorio GitHub con código fuente terminado y fácil de revisar.	2	X	2	
<b>2. Commits</b>	Hay capturas de pantalla de los commits más importantes con sus explicaciones detalladas. (El profesor puede preguntar para refrendar calificación).	4	X	4	
<b>3. Código fuente</b>	Hay porciones de código fuente importantes con numeración y explicaciones detalladas de sus funciones.	2	X	2	
<b>4. Ejecución</b>	Se incluyen ejecuciones/pruebas del código fuente explicadas gradualmente.	2	X	2	
<b>5. Pregunta</b>	Se responde con completitud a la pregunta formulada en la tarea. (El profesor puede preguntar para refrendar calificación).	2	X	2	
<b>6. Fechas</b>	Las fechas de modificación del código fuente están dentro de los plazos de fecha de entrega establecidos.	2	X	2	
<b>7. Ortografía</b>	El documento no muestra errores ortográficos.	2	X	2	
<b>8. Madurez</b>	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	4	
<b>Total</b>		20		20	

## 12. Referencias

- <https://www.python.org/>
- <https://www.pygame.org/news>
- <https://github.com/rescobedoq/pw2/tree/main/labs/lab04/Tarea-del-Ajedrez>