

# Informe de Laboratorio 05

## Tema: Python

Nota

Estudiante	Escuela	Asignatura
Rafael Diego Nina Calizaya rminacal@unsa.edu.pe	Escuela Profesional de Ingeniería de Sistemas	Programación Web 2 Semestre: III Código: 1702122

Laboratorio	Tema	Duración
05	Python	06

Semestre académico	Fecha de inicio	Fecha de entrega
2024-A	27 Mayo 2024	31 Mayo 2024

### 1. URL del Repositorio:

[https://github.com/DrN25/pw2\\_24a/tree/main/Lab05](https://github.com/DrN25/pw2_24a/tree/main/Lab05)

### 2. Tarea:

En esta tarea usted pondrá en práctica sus conocimientos de programación en Python para dibujar un tablero de Ajedrez.

#### Implementación de los métodos de la clase Picture.

Se implementaron los métodos de Picture, con los procedimientos y métodos pedidos en la practica.

- **verticalMirror():** Este método retorna una copia vertical de un Picture. Ya viene implementado previamente desde el repositorio de la práctica.

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

- **horizontalMirror():** Este método retorna una copia horizontal de un Picture. Aquí solo se invierte el orden del arreglo para voltear la imagen.

```
def horizontalMirror(self):
    """ Devuelve el espejo horizontal de la imagen """
    return Picture(self.img[::-1])
```

- **negative():** Este método retorna un Picture con sus colores opuestos. Aquí se crea un arreglo de valores opuestos llamado **inverter**, obteniendo los de la clase **Color**, para luego iterar sobre cada fila y columna de la imagen, donde se invertirá los caracteres según el arreglo **inverter**, y en caso de que sea " " se mantendrá igual.

```
def negative(self):
    """ Devuelve un negativo de la imagen """
    inverter = {'_': '=', '=': '_', '.': '@', '@': '.'}
    negative = []
    for row in self.img:
        negativeRow = []
        for value in row:
            negativeRow.append(inverter.get(value, value))
        negative.append(negativeRow)
    return Picture(negative)
```

- **join():** Este método retorna un nuevo Picture, donde Picture p está al lado derecho del Picture actual. Para conseguir ello, se itera sobre cada fila de Picture, donde en el nuevo arreglo se irán agregando cada fila del Picture actual más cada fila del Picture p.

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

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

- **up():** Este método retorna un nuevo Picture, donde el Picture actual está arriba de Picture p. Aquí solo se suman los arreglos para conseguir ello.

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

- **under():** Este método retorna un Picture, donde Picture p se sobrepone en el Picture actual. Aquí se itera sobre cada fila y columna de los arreglos, donde cada carácter de Picture p se colocara en el nuevo arreglo, a excepción de que el carácter sea igual a , donde se pondrá el carácter del Picture actual correspondiente a la fila y columna de ese momento.

```
def under(self, p):
    """ Devuelve una nueva figura poniendo la figura p sobre la
        figura actual """
    under = []
    for row in range(len(p.img)):
        rowTemp = ''
        for col in range(len(p.img[0])):
            if p.img[row][col] == ' ':
                rowTemp += self.img[row][col]
            else:
                rowTemp += p.img[row][col]
        under.append(rowTemp)
    return Picture(under)
```

- **horizontalRepeat():** Este método retorna un Picture que se repite n veces de manera horizontal. Para ello se itera sobre cada índice del arreglo actual, y se multiplica cada fila por la cantidad n entregada para dar como resultado las repeticiones horizontales.

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

- **verticalRepeat():** Este método retorna un Picture que se repite n veces de manera vertical. Para ello, se trabaja con un ciclo for, donde en cada ciclo se agregara la imagen actual al nuevo arreglo, y esto se repetirá n veces para dar como resultado las repeticiones verticales.

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

- **EXTRA: rotate():** Esta función retorna el Picture actual rotado 90° a la derecha. Aquí mediante 2 ciclos for se itero sobre la imagen actual. utilizando la función **reversed()** para iterar sobre las filas de abajo hacia arriba, pudiendo construir la imagen rotada 90°.

```
def rotate(self):  
    """Devuelve una figura rotada en 90 grados, puede ser en sentido horario  
    o antihorario"""  
    rotated = []  
    for col in range(len(self.img[0])):  
        rowTemp = ''  
        for row in reversed(range(len(self.img))):  
            rowTemp += self.img[row][col]  
        rotated.append(rowTemp)  
    return Picture(rotated)
```

## Ejercicios Resueltos.

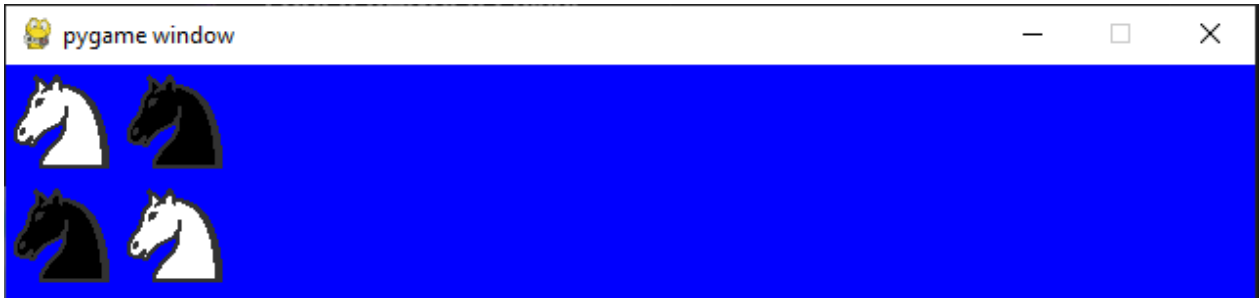
- **EJERCICIO A:** Se hizo uso de los métodos **negative()**, **join()** y **up()** para unir los caballos y cambiarlos de color.

```
knightWhite = knight
knightBlack = knight.negative()

row1 = knightWhite.join(knightBlack)
row2 = row1.negative()

total = row2.up(row1)

draw(total)
```



- **EJERCICIO B:** Se hizo uso de lo trabajado en el ejercicio 1 y se usó el método `verticalMirror()`.

```
knightWhite = knight
knightBlack = knight.negative()

row1 = knightWhite.join(knightBlack)
row2 = row1.verticalMirror()

total = row2.up(row1)

draw(total)
```



- **EJERCICIO C:** Se hizo uso del método `horizontalRepeat()` para crear 4 reinas una al lado de la otra.

```
queenWhite = queen
row1 = queen.horizontalRepeat(4)

total = row1

draw(total)
```



- **EJERCICIO D:** Se hizo uso de los métodos `negative()`, `join()` y `horizontalRepeat()`, buscando reutilizar el objeto `Picture` creado al inicio y no crear innecesariamente otros varios más.

```
squareWhite = square
squareBlack = squareWhite.negative()

p1 = squareWhite.join(squareBlack)
row1 = p1.horizontalRepeat(4)

total = row1

draw(total)
```



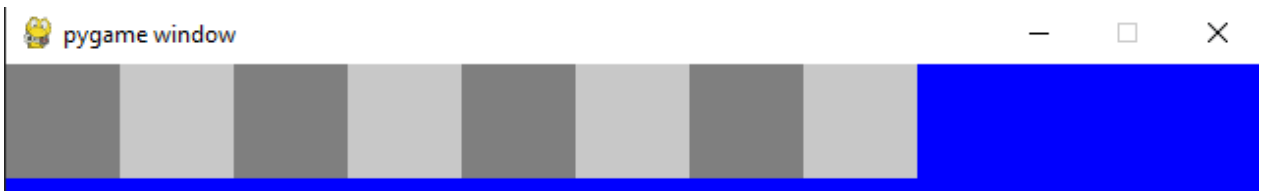
- **EJERCICIO E:** Se utilizó la solución del ejercicio 4 pero se le agrego el método `negative()` para obtener el resultado buscado.

```
squareWhite = square
squareBlack = squareWhite.negative()

p1 = squareWhite.join(squareBlack)
row1 = p1.horizontalRepeat(4)

total = row1.negative()

draw(total)
```



- **EJERCICIO F:** Se reutilizo la implementación de los ejercicios 4 y 5, además se utilizó los métodos `up()` y `verticalRepeat()`.

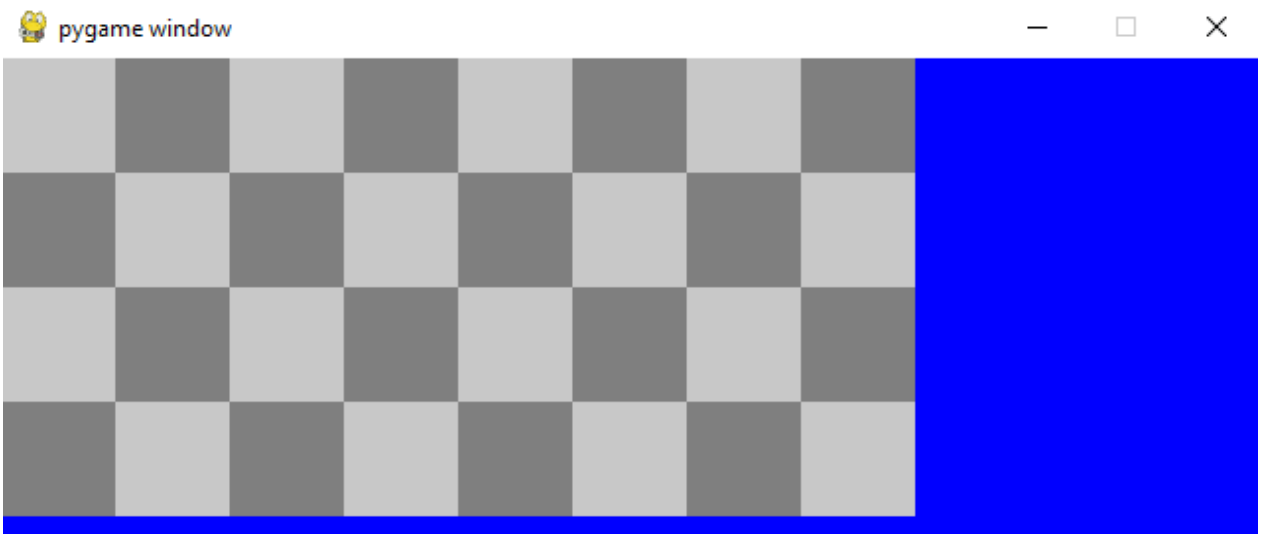
```
squareWhite = square
squareBlack = squareWhite.negative()

p1 = squareWhite.join(squareBlack)
row1 = p1.horizontalRepeat(4)
row2 = row1.negative()

p2 = row2.up(row1)

total = p2.verticalRepeat(2)

draw(total)
```



- **EJERCICIO G:** Se dividió el tablero en pequeñas subpartes las cuales se pudieran repetir o fueran el inverso de otras, por lo que se busco crear estas subpartes para al final unir las y acomodarlas con los métodos `join()`, `negative()`, `up()`, `verticalRepeat()`, `horizontalRepeat()` y `under()`. Con esto se logró completar el tablero de ajedrez.

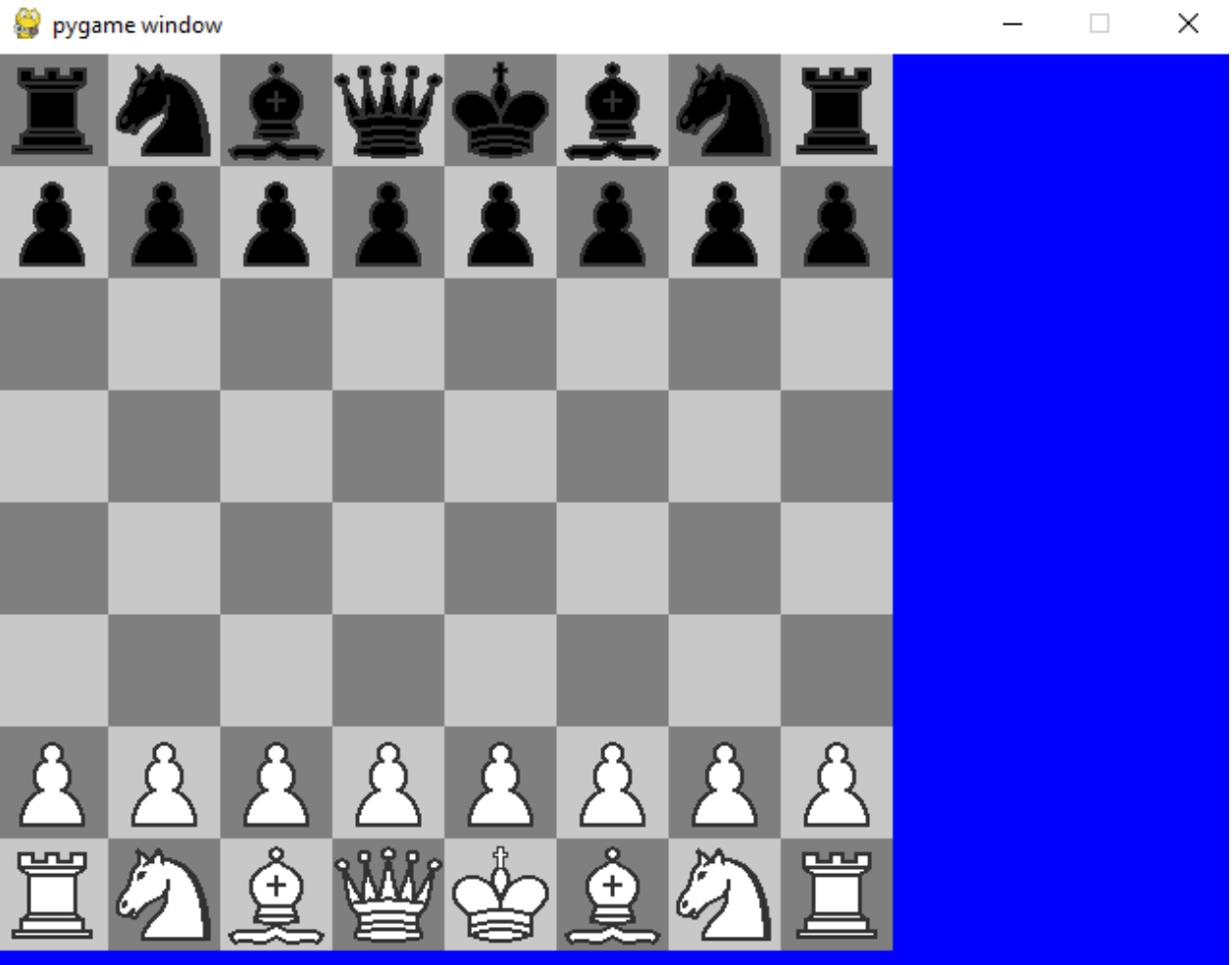
```
p1 = rock.join(knight.join(bishop.join(queen.join(king).join(bishop.join(knight.join(rock))))))
p2 = pawn.horizontalRepeat(8)
p3 = square.join(square.negative())

rowEmpty = p3.horizontalRepeat(4)
row1 = rowEmpty.under(p1)
row2 = rowEmpty.negative().under(p2)

p4 = row1.up(row2)
p5 = rowEmpty.up(rowEmpty.negative()).verticalRepeat(2)
p6 = row2.up(row1).negative()

total = p4.up(p5.up(p6))

draw(total)
```



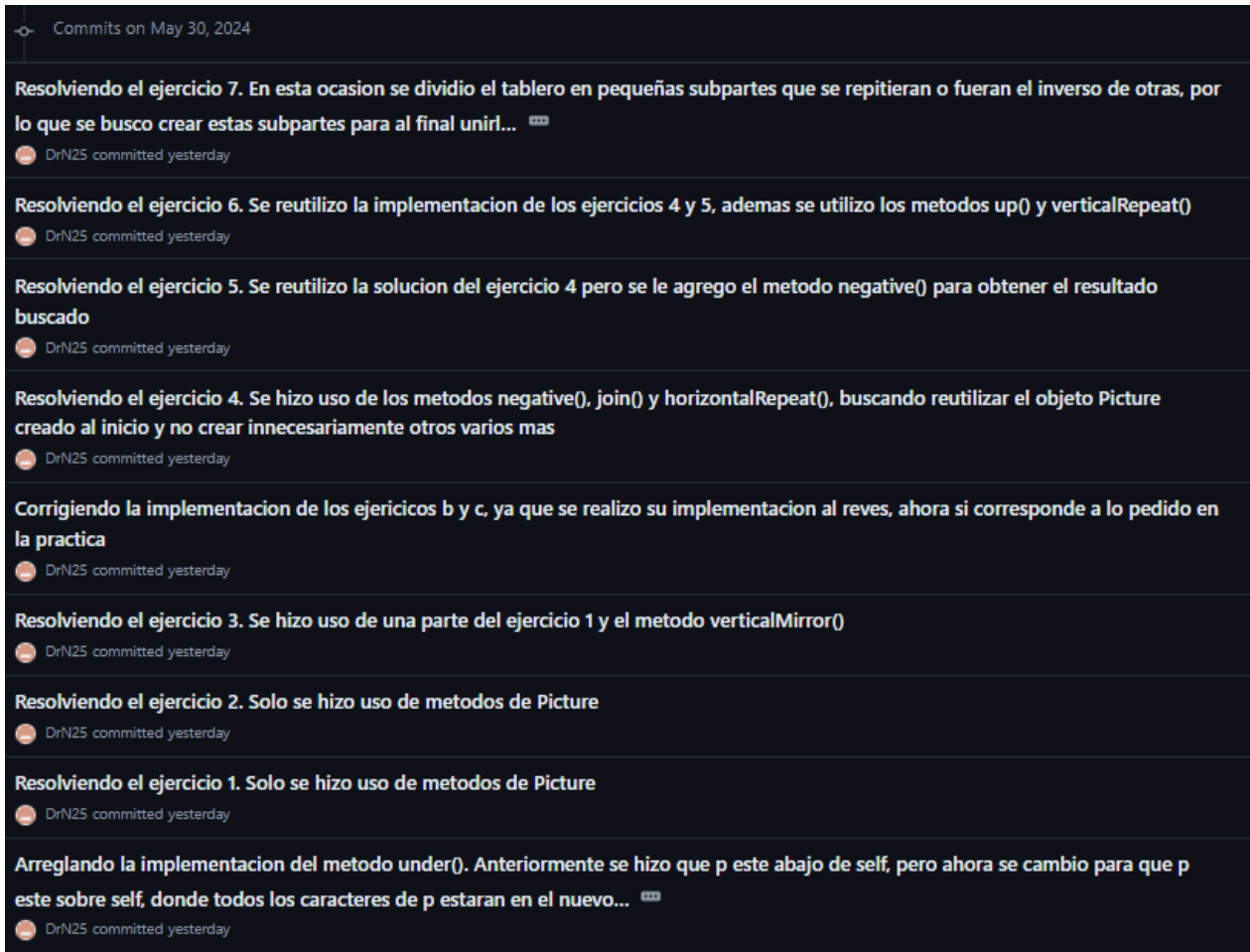


## Pregunta:


**Explique: ¿Para qué sirve el directorio pycache?**

Este directorio sirve para almacenar archivos caché de byte-code generados por Python. Gracias a ello se puede optimizar en el tiempo de carga de los módulos al evitar la recompilación del código fuente cada vez que se importa un archivo .py, mejorando el rendimiento y velocidad de ejecución de los programas en Python, siendo de utilidad para proyectos a gran escala con muchos módulos de por medio.


## 3. Commits realizados:




**EXTRA: Se implemento el metodo rotate(). Se creo un nuevo arreglo para retornar, y mediante 2 ciclos for se itero sobre la img actual. utilizandose la funcion reversed() para iterar sobre las filas...** xxx

 DrN25 committed yesterday


**Implementando los metodos verticalRepeat() y horizontalRepeat(). En ambos se creo un nuevo arreglo, en el 1ro se agregaban todas las filas de selg.img n veces, y en el 2do cada fila se repetia n ve...** xxx


 DrN25 committed yesterday

**Implementando los metodos up() y under(). Estos sumaran los arreglos de los 2 objetos Picture, en el 1ro se suma p + self y en el 2do self + p, funcionando de manera correcta de momento**


 DrN25 committed yesterday

**Implementando el metodo join(). Este recorrera cada indice de la imagen, donde en cada una sumara el actual con el entregado, de esta manera el entregado quedara a la derecha del actual. Tambien se...** xxx


 DrN25 committed 2 days ago

 Commits on May 29, 2024

**Implementando el metodo negative(). Este metodo evalua cada elemento del arreglo, donde con ayuda de otro arreglo llamado inverter se podra cambiar los caracteres por su inverso. Funciona correctam...** xxx

 DrN25 committed 2 days ago

**Trabajando los primeros metodos de la clase Picture, el metodo verticalMirror() funciona y se implemento el metodo horizontalMirror(), ambos mostrando resultados correctos de momento**

 DrN25 committed 2 days ago

## 4. Rúbrica:

**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	3	
<b>3. Código fuente</b>	Hay porciones de código fuente importantes con numeración y explicaciones detalladas de sus funciones.	2	X	1.5	
<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	3	
<b>TOTAL</b>		<b>20</b>		<b>17.5</b>	