

Informe de Laboratorio 04

Tema: Python

Nota

Estudiante	Escuela	Asignatura
Sebastian Arley Chirinos Negrón schirinosne@unsa.edu.pe	Escuela Profesional de Ingeniería de Sistemas	Programación Web 2 Semestre: I Código: 1702122

Laboratorio	Tema	Duración
04	Python	04 horas

Semestre académico	Fecha de inicio	Fecha de entrega
2023 - A	Del 29 Mayo 2023	Al 05 Junio 2023

1. Tarea

- En esta tarea 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:

2. Equipos, materiales y temas utilizados

- Programar usando Python.
- Mostrar un ejemplo de separación de intereses en clases: el modelo (lista de strings) de su vista (dibujo de gráficos).
- Listas
- Ciclos
- Programación orientada a objetos
- ¿Programación funcional?

3. URL de Repositorio Github

- URL del Repositorio GitHub para clonar o recuperar.
- <https://github.com/BastleyNait/PWEB02-B.git>
- URL para el laboratorio 04 en el Repositorio GitHub.
- <https://github.com/BastleyNait/PWEB02-B/tree/main/lab04>

4. Actividades con el repositorio GitHub

4.1. Ejercicio 01

- 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.

Listing 1: picture.py

```
1 from colors import *
2
3
4 class Picture:
5     def __init__(self, img):
6         self.img = img
7
8     def __eq__(self, other):
9         return self.img == other.img
10
11     def _invColor(self, color):
12         if color not in inverter:
13             return color
14         return inverter[color]
15
16     def verticalMirror(self):
17         """ Devuelve el espejo vertical de la imagen """
18         vertical = []
19         for value in self.img:
20             vertical.append(value[::-1])
21         return Picture(vertical)
22
23     def horizontalMirror(self):
24         """ Devuelve el espejo horizontal de la imagen """
25         return Picture(None)
26
27     def negative(self):
28         """ Devuelve un negativo de la imagen """
29         invertido = []
30         for elem in self.img:
31             aux = elem.replace("_", "=").replace(".", "@").replace("-", "=")
32             invertido.append(aux)
33         return Picture(invertido)
34
35     def join(self, p):
36         """ Devuelve una nueva figura poniendo la figura del argumento
37             al lado derecho de la figura actual """
```

```
38     lista_combinada = [x + y for x, y in zip(self.img, p.img)]
39     return Picture(lista_combinada)
40
41     def up(self, p):
42         return Picture(self.img + p.img)
43
44     def under(self, p):
45         """ Devuelve una nueva figura poniendo la figura p sobre la
46             figura actual """
47         superpuesto=[]
48         if "_" in p.img[0]:
49             for elem in self.img:
50                 aux = elem.replace(" ", "_")
51                 superpuesto.append(aux)
52         else:
53             for elem in self.img:
54                 aux = elem.replace(" ", "=")
55                 superpuesto.append(aux)
56         return Picture(superpuesto)
57
58     def horizontalRepeat(self, n):
59         """ Devuelve una nueva figura repitiendo la figura actual al costado
60             la cantidad de veces que indique el valor de n """
61         lista_combinada=[c * n for c in self.img]
62         return Picture(lista_combinada)
63
64
65     def verticalRepeat(self, n):
66         repeted = self.img * n
67         return Picture(repeted)
```

4.2. Ejercicio 02a

- Antes de ejecutar el código tenemos que tener en cuenta el importar la función draw del archivo interpreter.py y también todas las funciones de chessPicture:

Listing 2: Ejercicio2a.py

```
1 from interpreter import draw
2 from chessPictures import *
3 #se junta caballo blanco con negro se pone encima de caballo negro con blanco juntos
4 draw(knight.join(knight.negative()).up(knight.negative().join(knight)))
```

- Mostrando la Ejecución del código:



4.3. Ejercicio 02b

- Antes de ejecutar el código tenemos que tener en cuenta el importar la función draw del archivo interpreter.py y también todas las funciones de chessPicture:

Listing 3: Ejercicio2b.py

```
1 from interpreter import draw
2 from chessPictures import *
3 #Knight inverted
4 knightI = knight.verticalMirror()
5 draw(knight.join(knight.negative()).up(knightI.negative().join(knightI)))
```

- Mostrando la Ejecución del código:



4.4. Ejercicio 02c

- Antes de ejecutar el código tenemos que tener en cuenta el importar la función draw del archivo interpreter.py y también todas las funciones de chessPicture:

Listing 4: Ejercicio2c.py

```
1 from interpreter import draw
2 from chessPictures import *
3
4 draw(queen.join(queen.join(queen.join(queen))))
```

- Mostrando la Ejecución del código:



4.5. Ejercicio 02d

- Antes de ejecutar el código tenemos que tener en cuenta el importar la función draw del archivo interpreter.py y también todas las funciones de chessPicture:

Listing 5: Ejercicio2d.py

```
1 from interpreter import draw
2 from chessPictures import *
3 squarex2 = square.join(square.negative())
4 draw(squarex2.join(squarex2.join(squarex2.join(squarex2))))
```

- Mostrando la Ejecución del código:



4.6. Ejercicio 02d

- Antes de ejecutar el código tenemos que tener en cuenta el importar la función draw del archivo interpreter.py y también todas las funciones de chessPicture:

Listing 6: Ejercicio2d.py

```
1 from interpreter import draw
2 from chessPictures import *
3 squarex2 = square.join(square.negative())
4 draw(squarex2.join(squarex2.join(squarex2.join(squarex2))))
```

- Mostrando la Ejecución del código:



4.7. Ejercicio 02e

- Antes de ejecutar el código tenemos que tener en cuenta el importar la función draw del archivo interpreter.py y también todas las funciones de chessPicture:

Listing 7: Ejercicio2e.py

```
1 from interpreter import draw
2 from chessPictures import *
3 squarex2 = square.negative().join(square)
4 draw(squarex2.join(squarex2.join(squarex2.join(squarex2))))
```

- Mostrando la Ejecución del código:



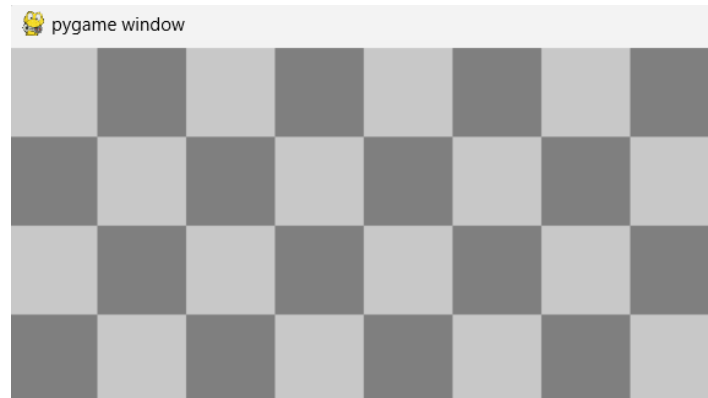
4.8. Ejercicio 02f

- Antes de ejecutar el código tenemos que tener en cuenta el importar la función draw del archivo interpreter.py y también todas las funciones de chessPicture:

Listing 8: Ejercicio2f.py

```
1 from interpreter import draw
2 from chessPictures import *
3 squareN = square.negative().join(square)
4 squareB = square.join(square.negative())
5
6 blanco = squareB.join(squareB.join(squareB.join(squareB)))
7 negro = squareN.join(squareN.join(squareN.join(squareN)))
8 draw(blanco.up(negro).up(blanco.up(negro)))
```

- Mostrando la Ejecución del código:



4.9. Ejercicio 02g

- Antes de ejecutar el código tenemos que tener en cuenta el importar la función draw del archivo interpreter.py y también todas las funciones de chessPicture:

Listing 9: Ejercicio2g.py

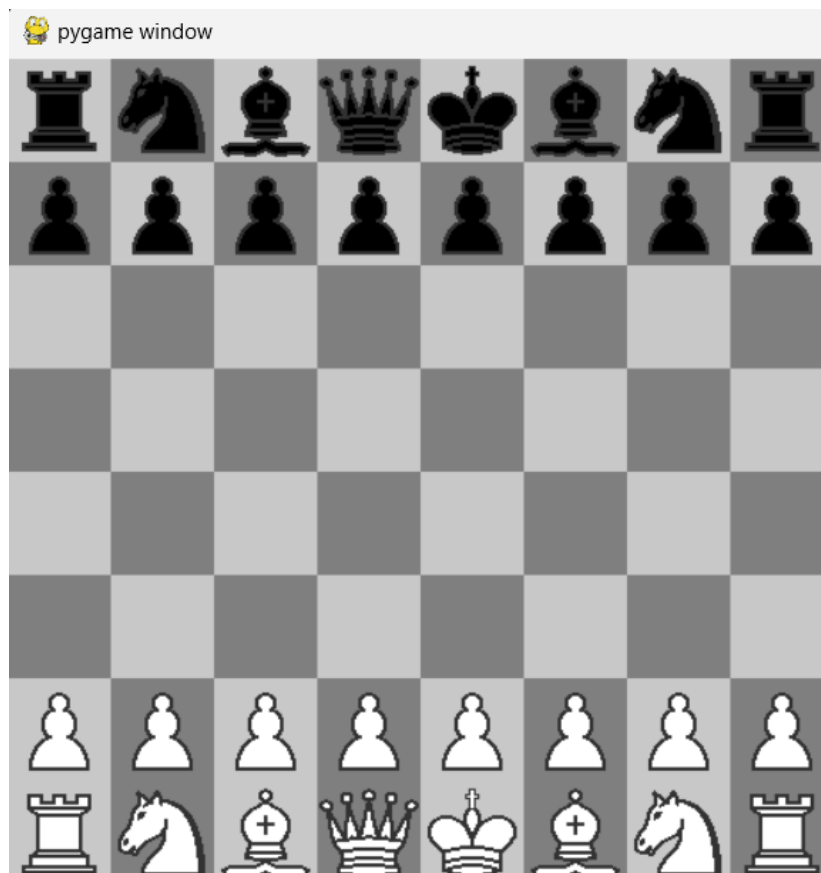
```

1 from interpreter import draw
2 from chessPictures import *
3 """PRUEBAS"""
4 # squareN = square.negative().join(square)
5 # squareB = square.join(square.negative())
6
7 # blanco = squareB.join(squareB.join(squareB.join(squareB)))
8 # negro = squareN.join(squareN.join(squareN.join(squareN)))
9
10 # draw(blanco.up(negro).verticalRepeat(4))
11
12 # draw(knight.negative().under(square.negative()))
13
14 # draw(knight.horizontalRepeat(6))
15 """FILA PEONES"""
16 # doble peon blanco sobre tablero blanco y negro x 6
17 peonx2B = (pawn.under(square).join(pawn.under(square.negative()))).horizontalRepeat(4)
18 # doble peon negro sobre tablero negro y blanco x 6
19 peonx2N =
20     (pawn.negative().under(square.negative()).join(pawn.negative().under(square))).horizontalRepeat(4)
21 """TABLERO DEL MEDIO"""
22 # tablero 2 x 2
23 tab2x2 = ((square.join(square.negative())).up(square.negative().join(square)))
24 # tablero 8 x 4
25 tab8x4 = (tab2x2.horizontalRepeat(4)).verticalRepeat(2)
26 # Haciendo fichas con sus colores de tableros
27 """TORRE"""
28 # torre negra tablero blanco
29 rockNtB = rock.negative().under(square)
30 # torre negra tablero negro
31 rockNtN = rock.negative().under(square.negative())
32 # torre blanca tablero negro
33 rockBtN = rock.under(square.negative())
34 # torre blanca tablero blanco

```

```
34 rockBtB = rock.under(square)
35 """CABALLO"""
36 # torre negra tablero blanco
37 knightNtB = knight.negative().under(square)
38 # torre negra tablero negro
39 knightNtN = knight.negative().under(square.negative())
40 # torre blanca tablero negro
41 knightBtN = knight.under(square.negative())
42 # torre blanca tablero blanco
43 knightBtB = knight.under(square)
44 """ALFIL"""
45 # torre negra tablero blanco
46 bishopNtB = bishop.negative().under(square)
47 # torre negra tablero negro
48 bishopNtN = bishop.negative().under(square.negative())
49 # torre blanca tablero negro
50 bishopBtN = bishop.under(square.negative())
51 # torre blanca tablero blanco
52 bishopBtB = bishop.under(square)
53 """REINA"""
54 # torre negra tablero blanco
55 queenNtB = queen.negative().under(square)
56 # torre negra tablero negro
57 queenNtN = queen.negative().under(square.negative())
58 # torre blanca tablero negro
59 queenBtN = queen.under(square.negative())
60 # torre blanca tablero blanco
61 queenBtB = queen.under(square)
62
63 """REY"""
64 # torre negra tablero blanco
65 kingNtB = king.negative().under(square)
66 # torre negra tablero negro
67 kingNtN = king.negative().under(square.negative())
68 # torre blanca tablero negro
69 kingBtN = king.under(square.negative())
70 # torre blanca tablero blanco
71 kingBtB = king.under(square)
72
73 """CADENA NEGRO"""
74 cadenaN =
75     rockNtB.join(knightNtN.join(bishopNtB.join(queenNtN.join(kingNtB.join(bishopNtN.join(knightNtB.join(rockNtN)
76 """CADENA BLANCO"""
77 cadenaB =
78     rockBtB.join(knightBtN.join(bishopBtB.join(queenBtN.join(kingBtB.join(bishopBtN.join(knightBtB.join(rockBtN)
79 """DIBUJANDO TABLERO COMPLETO"""
80 draw(((cadenaN.up(peonx2N)).up(tab8x4)).up(peonx2B.up(cadenaB)))
```

- Motrando la Ejecución del codigo:



4.10. Estructura de laboratorio 04

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

```
lab04/
+---EjerciciosDocente
|   defs.py
|   esEscalar.py
|   esPalindromo.py
|   esUnitaria.py
|   numeroPares.py
|   operadoresArit.py
|   pythonClass.py
|   strings.py
|   tablaDeMulti.py
|   test_esEscalar.py
|   test_esUnitaria.py
|   tiposDeDatos.py
|
+---Latex
|   |   .gitignore
|   |   Pweb02_lab04_schirinosne.pdf
|   |   Pweb02_lab04_schirinosne.tex
|   |
|   +---img
```

```
| | Ejercicio2a.png
| | Ejercicio2b.png
| | Ejercicio2c.png
| | Ejercicio2d.png
| | Ejercicio2e.png
| | Ejercicio2f.png
| | Ejercicio2g.png
| | logo_abet.png
| | logo_episunsa.png
| | logo_unsa.jpg
| | pseudocodigo_insercion.png
| |
| | \---src
| | Ejercicio2a.py
| | Ejercicio2b.py
| | Ejercicio2c.py
| | Ejercicio2d.py
| | Ejercicio2e.py
| | Ejercicio2f.py
| | Ejercicio2g.py
| | Insertion01.java
| | picture.py
| |
| | \---Tarea-del-Ajedrez
| | .gitignore
| | chessPictures.py
| | colors.py
| | Ejercicio2a.py
| | Ejercicio2b.py
| | Ejercicio2c.py
| | Ejercicio2d.py
| | Ejercicio2e.py
| | Ejercicio2f.py
| | Ejercicio2g.py
| | interpreter.py
| | picture.py
| | pieces.py
| | prueba.py
```

5. Pregunta: ¿Qué son los archivos *.pyc?

- Los archivos .pyc son archivos de código compilado en Python. Cuando un archivo fuente de Python (.py) se ejecuta, el intérprete de Python compila ese código en bytecode, que es una representación intermedia del código que puede ser ejecutada más rápido por la máquina virtual de Python. Los archivos *.pyc contienen este bytecode compilado y se generan automáticamente cuando se importa un módulo en Python.

6. Pregunta: ¿Para qué sirve el directorio pycache?

- El directorio "pycache" es un directorio que se crea automáticamente en Python 3 para almacenar los archivos *.pyc. Cuando se importa un módulo en Python, el intérprete buscará si existe un archivo *.pyc correspondiente en el directorio "pycache". Si lo encuentra y es más reciente que el archivo *.py fuente, el intérprete utilizará el archivo *.pyc en su lugar para ahorrar tiempo

de compilación. Si no existe un archivo *.pyc o está desactualizado, el intérprete generará uno nuevo.

7. Pregunta: ¿Cuáles son los usos y lo que representa el subguión en Python?

- En cuanto al subguión en Python, se le conoce como underscore y se utiliza de diferentes formas:
- Nombres de variables especiales: En Python, el subguión se utiliza para nombres de variables especiales que tienen un significado específico. Por ejemplo, un subguión simple se utiliza a menudo como un nombre de variable temporal o como un lugar para ignorar valores que no se necesitan.
- Convención para nombres privados: El subguión doble al inicio de un nombre de variable por ejemplo, nombre se utiliza como convención para indicar que un atributo o método es "privado".^{en} Python. No hay verdaderos atributos o métodos privados en Python, pero se considera una convención de estilo no acceder directamente a estos atributos o métodos desde fuera de la clase.
- Uso en importaciones: El subguión se utiliza a menudo en las importaciones de módulos en Python.

8. Referencias

- https://www.w3schools.com/python/python_reference.asp
- <https://docs.python.org/3/tutorial/>