# Informe de Laboratorio 04

## Tema: Python

| Nota |
| --- |
| |

| Estudiante | Escuela | Asignatura |
| --- | --- | --- |
| Sebastian Arley Chirinos Negrón<br>schirinosne@unsa.edu.pe | Escuela Profesional de Ingeniería de Sistemas | Programación Web 2<br>Semestre: I<br>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

```python
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]

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

    def horizontalMirror(self):
        """ Devuelve el espejo horizontal de la imagen """
        return Picture(None)

    def negative(self):
        """ Devuelve un negativo de la imagen """
        invertido = []
        for elem in self.img:
            aux = elem.replace("_", "=").replace(".", "@").replace("_", "=")
            invertido.append(aux)
        return Picture(invertido)

    def join(self, p):
        """ Devuelve una nueva figura poniendo la figura del argumento
            al lado derecho de la figura actual """
```

```
        lista_combinada = [x + y for x, y in zip(self.img, p.img)]
        return Picture(lista_combinada)

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

    def under(self, p):
        """ Devuelve una nueva figura poniendo la figura p sobre la
            figura actual """
        superpuesto=[]
        if "_" in p.img[0]:
            for elem in self.img:
                aux = elem.replace(" ", "_")
                superpuesto.append(aux)
        else:
            for elem in self.img:
                aux = elem.replace(" ", "=")
                superpuesto.append(aux)
        return Picture(superpuesto)

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


    def verticalRepeat(self, n):
        repeted = self.img * n
        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

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

- Motrando la Ejecución del codigo:


## 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

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

- Motrando la Ejecución del codigo:


## 4.4. Ejercicio 02c

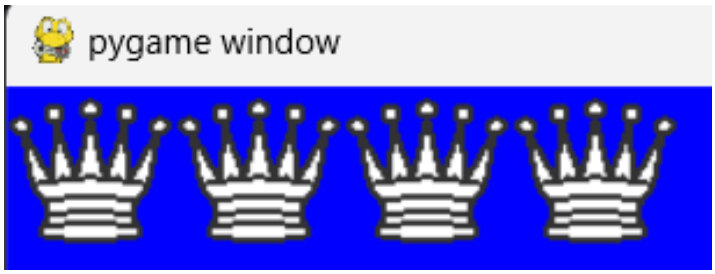
- Antes de ejecutar el código tenemos que tener en cuenta el importar la función draw del archivo interpreter.py y también todas las funciones de chessPicture:

Listing 4: Ejercicio2c.py

```
from interpreter import draw
from chessPictures import *

draw(queen.join(queen.join(queen.join(queen))))
```

- Motrando la Ejecución del codigo:



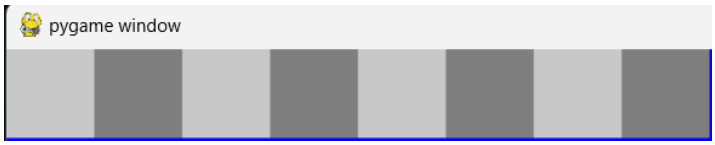

## 4.5. Ejercicio 02d

- 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

```
from interpreter import draw
from chessPictures import *
squarex2 = square.join(square.negative())
draw(squarex2.join(squarex2.join(squarex2.join(squarex2))))
```

- Motrando la Ejecución del codigo:




## 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

```
from interpreter import draw
from chessPictures import *
squarex2 = square.join(square.negative())
draw(squarex2.join(squarex2.join(squarex2.join(squarex2))))
```

- Motrando la Ejecución del codigo:


## 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

```
from interpreter import draw
from chessPictures import *
squarex2 = square.negative().join(square)
draw(squarex2.join(squarex2.join(squarex2.join(squarex2))))
```

- Motrando la Ejecución del codigo:


## 4.8. Ejercicio 02f

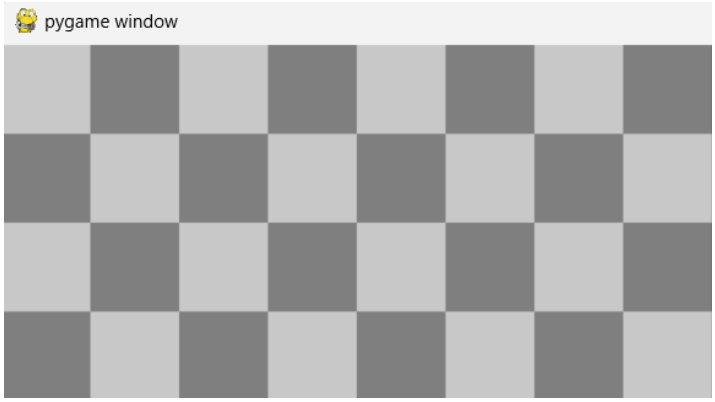
- Antes de ejecutar el código tenemos que tener en cuenta el importar la función draw del archivo interpreter.py y también todas las funciones de chessPicture:

Listing 8: Ejercicio2f.py

```
from interpreter import draw
from chessPictures import *
squareN = square.negative().join(square)
squareB = square.join(square.negative())

blanco = squareB.join(squareB.join(squareB.join(squareB)))
negro = squareN.join(squareN.join(squareN.join(squareN)))
draw(blanco.up(negro).up(blanco.up(negro)))
```

- Motrando la Ejecución del codigo:




## 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

```
from interpreter import draw
from chessPictures import *
"""PRUEBAS"""
# squareN = square.negative().join(square)
# squareB = square.join(square.negative())

# blanco = squareB.join(squareB.join(squareB.join(squareB)))
# negro = squareN.join(squareN.join(squareN.join(squareN)))

# draw(blanco.up(negro).verticalRepeat(4))

# draw(knight.negative().under(square.negative()))

# draw(knight.horizontalRepeat(6))
"""FILA PEONES"""
# doble peon blanco sobre tablero blanco y negro x 6
peonx2B = (pawn.under(square).join(pawn.under(square.negative()))).horizontalRepeat(4)
# doble peon negro sobre tablero negro y blanco x 6
peonx2N =
    (pawn.negative().under(square.negative()).join(pawn.negative().under(square))).horizontalRepeat(4)
"""TABLERO DEL MEDIO"""
# tablero 2 x 2
tab2x2 = ((square.join(square.negative())).up(square.negative().join(square)))
# tablero 8 x 4
tab8x4 = (tab2x2.horizontalRepeat(4)).verticalRepeat(2)
# Haciendo fichas con sus coleres de tableros
"""TORRE"""
# torre negra tablero blanco
rockNtB = rock.negative().under(square)
# torre negra tablero negro
rockNtN = rock.negative().under(square.negative())
# torre blanca tablero negro
rockBtN = rock.under(square.negative())
# torre blanca tablero blanco
```

```
rockBtB = rock.under(square)
"""CABALLO"""
# torre negra tablero blanco
knightNtB = knight.negative().under(square)
# torre negra tablero negro
knightNtN = knight.negative().under(square.negative())
# torre blanca tablero negro
knightBtN = knight.under(square.negative())
# torre blanca tablero blanco
knightBtB = knight.under(square)
"""ALFIL"""
# torre negra tablero blanco
bishopNtB = bishop.negative().under(square)
# torre negra tablero negro
bishopNtN = bishop.negative().under(square.negative())
# torre blanca tablero negro
bishopBtN = bishop.under(square.negative())
# torre blanca tablero blanco
bishopBtB = bishop.under(square)
"""REINA"""
# torre negra tablero blanco
queenNtB = queen.negative().under(square)
# torre negra tablero negro
queenNtN = queen.negative().under(square.negative())
# torre blanca tablero negro
queenBtN = queen.under(square.negative())
# torre blanca tablero blanco
queenBtB = queen.under(square)

"""REY"""
# torre negra tablero blanco
kingNtB = king.negative().under(square)
# torre negra tablero negro
kingNtN = king.negative().under(square.negative())
# torre blanca tablero negro
kingBtN = king.under(square.negative())
# torre blanca tablero blanco
kingBtB = king.under(square)

"""CADENA NEGRO"""
cadenaN =
    rockNtB.join(knightNtN.join(bishopNtB.join(queenNtN.join(kingNtB.join(bishopNtN.join(knightNtB.join(rockNtN)

"""CADENA BLANCO"""
cadenaB =
    rockBtB.join(knightBtN.join(bishopBtB.join(queenBtN.join(kingBtB.join(bishopBtN.join(knightBtB.join(rockBtN)

"""DIBUJANDO TABLERO COMPLETO"""
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:

```
lab01/
|--- Insertion.java
|--- latex
   |--- img
   |   |--- logo_abet.png
   |   |--- logo_episunsa.png
   |   |--- logo_unsa.jpg
   |   |--- pseudocodigo_insercion.png
   |--- programacion_lab01_rescobedoq_v1.0.pdf
   |--- programacion_lab01_rescobedoq_v1.0.tex
   |--- src
      |--- Insertion01.java
```

## 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/`