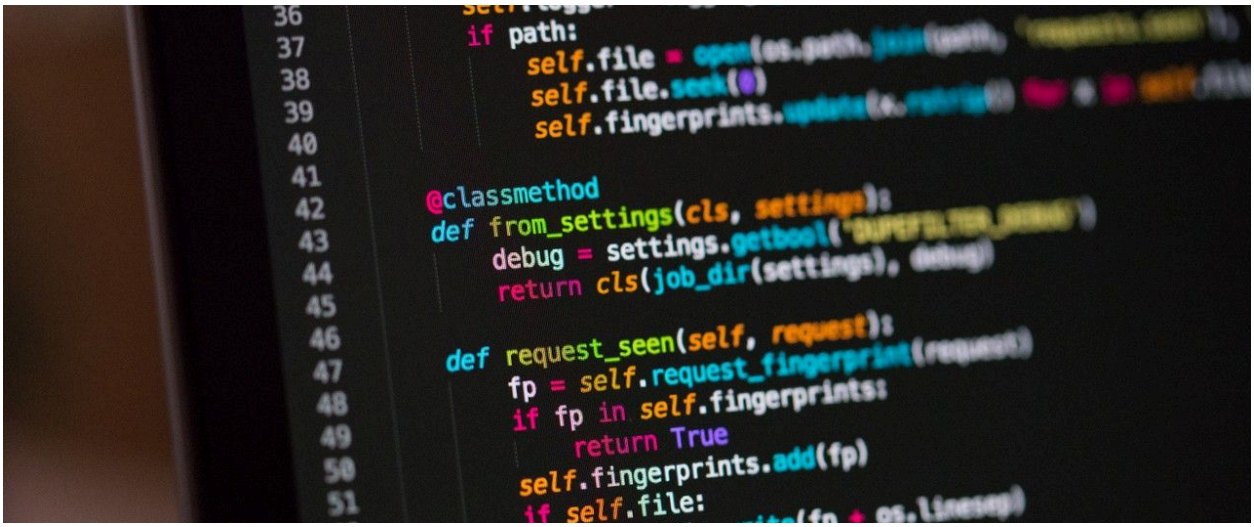

Documentación ejercicios con pilas

Ejercicio 1

Implementa un método que reciba una pila de enteros como único parámetro. Este método llamado “separarParImpar” deberá retornar la pila con los números pares en la parte inferior y los impares en la superior.

modulos.py:

```
def ordena(pila):
    lista_temp = []

    while pila:
        lista_temp.append(pila.pop()) # Sacamos con (pop) los elementos de la pila

    lista_temp.sort(reverse=True) # esto se utiliza para ordenar la pila

    pila_ordenada = [] # se crea una lista la cual almacenara ya los valores ordena
    for num in lista_temp:
        pila_ordenada.append(num) # se agregan los eleementos ordenados a la nu
```

```
    return pila_ordenada # retorna la pila ya ordenada de mayor a menor
```

pila.py:

```
class Pila:
    def __init__(self):
        self.elementos = []
```

Lista almacenara los elementos de la lista.

```
 def push(self, elemento):
        # Agrega un elemento a la cima de la pila
        self.elementos.append(elemento)
```

Para agregar un nuevo dato a la pila, se utiliza el método push, que simplemente añade el elemento al final de la lista interna. Esto equivale a colocarlo en la cima de la pila.

```
    def pop(self):
        return self.elementos.pop()
# Extrae y retorna el último elemento (LIFO)
```

```
  def esta_vacia(self):
        # Retorna True si la pila está vacía, False en caso contrario
        return len(self.elementos) == 0

    def __str__(self):
        # Representación en forma de lista para impresión
        return str(self.elementos)


def separarParImpar(pila_original):
```

```
    #creamos dos pilas auxiliares: una para pares y otra para impares
    pares = Pila()
    impares = Pila()

    #mientras haya elementos en la pila original, los extraemos uno a uno
    while not pila_original.esta_vacia():
        #extraemos el número de la cima de la pila
        numero = pila_original.pop()

        #si el número es par, lo agregamos a la pila de pares
        if numero % 2 == 0:
            pares.push(numero)
        #si el número es impar, lo agregamos a la pila de impares
        else:
            impares.push(numero)

    #creamos la pila resultado donde se unirán los pares primero, luego los impare
    resultado = Pila()

    # Primero agregamos los números pares a la pila resultado
    while not pares.esta_vacia():
        resultado.push(pares.pop())

    #luego agregamos los números impares a la pila resultado
    while not impares.esta_vacia():
        resultado.push(impares.pop())

    #retornamos la pila reconstruida con los pares en la cima y los impares debajo
    return resultado
```

principal.py

```
from Pila import Pila, separarParImpar
from colorama import Fore, init, Style
init(autoreset=True)
```

```
#Se le solicita al usuario que ingrese números separados por comas
entrada_usuario = input(Style.BRIGHT+"Ingresa números enteros separados por

#Convertimos la cadena en una lista de enteros, ignorando espacios en blanco
numeros = [int(n.strip()) for n in entrada_usuario.split(",") if n.strip().isdigit()]

#Creamos una pila y agregamos los números ingresados
entrada = Pila()
for numero in numeros:
    entrada.push(numero)

#Llamamos a la función que separa pares e impares
salida = separarParImpar(entrada)

print(Fore.YELLOW + "Entrada:", numeros)
print(Fore.GREEN + "Salida :", salida)
```

Ejercicio 2

Implementa un método llamado “ordena” que reciba una pila de enteros como parámetro y devuelva la pila ordenada de mayor (fondo de la pila) a menor (top de la pila).

main.py:

```
from modulos import ordena

# Entrada del usuario
entrada = input("Ingrese los números separados por espacios: ")

valores = []  # Lista para los números convertidos
numero = ""   # Acumulador de caracteres

# Recorremos cada carácter de la cadena
for caracter in entrada:
```

```
    if caracter != " ":
        numero += caracter  # Acumulamos el dígito
    else:
        if numero != "":
            valores.append(int(numero))  # Convertimos y guardamos
            numero = ""  # Reiniciamos para el siguiente número

# Agregamos el último número si no hay espacio al final
if numero != "":
    valores.append(int(numero))



# Creamos la pila original
pila = []

# Agregamos los valores a la pila
for num in valores:
    pila.append(num)

# Ordenamos la pila
resultado = ordena(pila)

# Mostramos el resultado final
print("Pila ordenada (de mayor a menor):", resultado)
```

modulos.py

```
def ordena(pila): # se proporciona el parametro pila
    lista_temp = [] # se crea una lista temporal donde se alamcenaran los valores i
    while pila:
        lista_temp.append(pila.pop()) # Sacamos con (pop) los elementos de la pila

    lista_temp.sort(reverse=True) # esto se utiliza para ordenar la pila

    pila_ordenada = [] # se crea una lista la cual almacenara ya los valores ordena
    for num in lista_temp:
```

```
        pila_ordenada.append(num) # se agregan los eleementos ordenados a la nu

    return pila_ordenada # retorna la pila ya ordenada de mayor a menor
```

Ejercicio 3

Diseñar un método “Convbinario” que reciba un entero como parámetro. La función, usando una pila, deberá mostrar el número en código binario.

```
from PilaBinaria import BinarioPila
import os
def limpiarPantalla():
    os.system("clear")
    os.system("cls")
def main():

    #Menu
    while True:
        limpiarPantalla()
        print("======       Ejercicio 3      ======")
        print("=== Conversor de Decimal a Binario ===")
        print("1. Convertir un número entero a binario")
        print("2. Salir")
        try:
            opc = int(input("Seleccione una opción: "))
        except ValueError:
            print("Ingrese un número válido.\n")
            continue

        match opc:
            case 1:
                # Crear una instancia de la clase BinarioPila
                binario_pila = BinarioPila()
```

```
            numero = int(input("Ingrese un número entero: "))

            resultado = binario_pila.convertir(numero)
            print(f"El número {numero} en binario es: {resultado}")
            input("Presione Enter para continuar...")
```

Cuando el usuario elige convertir un número (case 1), se crea una instancia de `BinarioPila`, se solicita el número entero y se llama al método `convertir()` para obtener su versión binaria. Luego, se muestra el resultado al usuario.

```
case 2:
            print("Saliendo del programa...")
            return
        case _:
            print("Opción no válida. Intente nuevamente.\n")

if __name__ == "__main__":
    main()
```

PilaBinaria.py

```
class Pila:
    def __init__(self, capacidad):
        self.capacidad = capacidad
        self.elementos = [None] * capacidad # Inicializa la pila con una capacidad fi
        self.tope = -1 # Índice del último elemento agregado
```

La clase Pila representa una pila con capacidad fija. Usa una lista predefinida y un índice tope que indica la cima de la pila. Al iniciar, tope está en -1, lo que indica

que está vacía.

```
def push(self, elemento):  # Agrega un elemento a la cima de la pila si hay espac
        if self.tope < self.capacidad - 1:
            self.tope += 1
            self.elementos[self.tope] = elemento
        else:
            print("Error: Pila llena. No se puede insertar.")
```

Este método agrega un elemento a la pila si aún no ha alcanzado su capacidad máxima. Si está llena, muestra un error.

```
  def is_empty(self): # Verifica si la pila está vacía
        return self.tope == -1
    def pop(self):
        if self.is_empty():
            print("Error: Pila vacía. No se puede eliminar.")
            return None
        eliminado = self.elementos[self.tope]
        self.elementos[self.tope] = None
        self.tope -= 1
        return eliminado
```

`pop()` extrae el elemento de la cima si la pila no está vacía. Si está vacía, devuelve None y muestra un mensaje.

```
class BinarioPila:
    def __init__(self, capacidad=32):
        self.pila = Pila(capacidad)
```

Esta clase contiene la lógica que convierte un número entero en binario usando la pila. Tiene un atributo pila, que es una instancia de la clase Pila, creada con una

capacidad predeterminada de 32 elementos.

```
def convertir(self, numero):
    if numero == 0:
        return "[0]"

    self.pila = Pila(self.pila.capacidad)  # Reiniciar pila por si se reutiliza

    while numero > 0:
        residuo = numero % 2
        self.pila.push(residuo)
        numero //= 2

    binario = ""
    while not self.pila.is_empty():
        binario += str(self.pila.pop())

    return f"[{binario}]"
```

Este método es el núcleo del programa. Si el número ingresado es 0, retorna inmediatamente `[0]`. Luego, mientras el número sea mayor que 0, obtiene el residuo de la división entre 2 (el bit menos significativo) y lo apila. Cuando termina, extrae los bits desde la pila en orden inverso, formando así el número binario.