1)

precios\_frutas = {'Banana': 1200, 'Ananá': 2500, 'Melón': 3000, 'Uva': 1450}

nuevas\_frutas = {

'Naranja': 1200,

'Manzana': 1500,

'Pera': 2300

}

precios\_frutas.update(nuevas\_frutas)

print("Diccionario actualizado:")

for fruta, precio in precios\_frutas.items():

print(f"{fruta}: ${precio}")

2)

precios\_frutas = {

'Banana': 1200,

'Ananá': 2500,

'Melón': 3000,

'Uva': 1450,

'Naranja': 1200,

'Manzana': 1500,

'Pera': 2300

}

precios\_frutas['Banana'] = 1330

precios\_frutas['Manzana'] = 1700

precios\_frutas['Melón'] = 2800

print("Precios actualizados:")

for fruta, precio in sorted(precios\_frutas.items()):

print(f"{fruta:.<15} ${precio:>5}") # Alineación con puntos y formato de precio

3)

precios\_frutas = {

'Banana': 1330,

'Ananá': 2500,

'Melón': 2800,

'Uva': 1450,

'Naranja': 1200,

'Manzana': 1700,

'Pera': 2300

}

lista\_frutas = list(precios\_frutas.keys())

print("Lista de frutas:", lista\_frutas)

4)

class Persona:

def \_\_init\_\_(self, nombre: str, pais: str, edad: int):

self.nombre = nombre

self.pais = pais

self.edad = edad

def saludar(self):

print(f"¡Hola! Soy {self.nombre}, vivo en {self.pais} y tengo {self.edad} años.")

5)

import math

class Circulo:

def \_\_init\_\_(self, radio: float):

self.radio = radio

def calcular\_area(self) -> float:

return math.pi \* self.radio \*\* 2

def calcular\_perimetro(self) -> float:

return 2 \* math.pi \* self.radio

6)

def balanceado(s):

stack = []

mapping = {')': '(', '}': '{', ']': '['}

for char in s:

if char in mapping.values():

stack.append(char)

elif char in mapping:

if not stack or stack[-1] != mapping[char]:

return False

stack.pop()

return len(stack) == 0

7)

from collections import deque

class Banco:

def \_\_init\_\_(self):

self.cola\_clientes = deque()

def agregar\_cliente(self, nombre\_cliente: str):

self.cola\_clientes.append(nombre\_cliente)

print(f"Cliente '{nombre\_cliente}' agregado a la cola.")

def atender\_cliente(self) -> str:

if not self.cola\_clientes:

raise ValueError("No hay clientes en la cola.")

cliente\_atendido = self.cola\_clientes.popleft()

print(f"Cliente '{cliente\_atendido}' atendido.")

return cliente\_atendido

def mostrar\_siguiente(self) -> str:

if not self.cola\_clientes:

raise ValueError("No hay clientes en la cola.")

siguiente = self.cola\_clientes[0]

print(f"Siguiente cliente en la fila: '{siguiente}'")

return siguiente

8)

class Nodo:

def \_\_init\_\_(self, dato):

self.dato = dato

self.siguiente = None

class ListaEnlazada:

def \_\_init\_\_(self):

self.cabeza = None

def insertar\_al\_inicio(self, dato):

nuevo\_nodo = Nodo(dato)

nuevo\_nodo.siguiente = self.cabeza

self.cabeza = nuevo\_nodo

def mostrar\_lista(self):

nodo\_actual = self.cabeza

while nodo\_actual is not None:

print(nodo\_actual.dato, end=" -> ")

nodo\_actual = nodo\_actual.siguiente

print("None")

9)

class Node:

def \_init\_(self, value):

self.value = value

self.next = None

def invertir\_lista(head):

prev = None

current = head

while current:

next\_temp = current.next

current.next = prev

prev = current

current = next\_temp

return prev