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
```