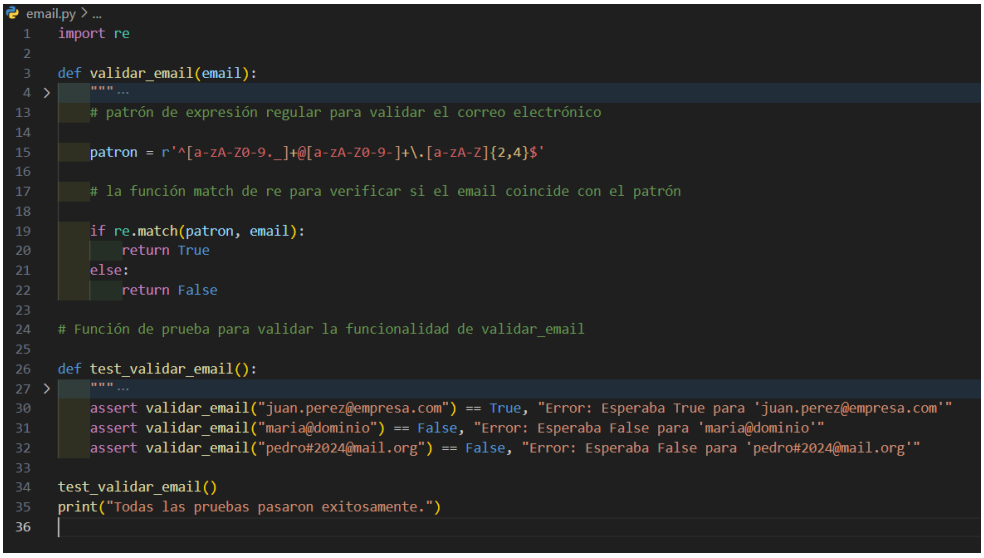
## Análisis Taller # 2

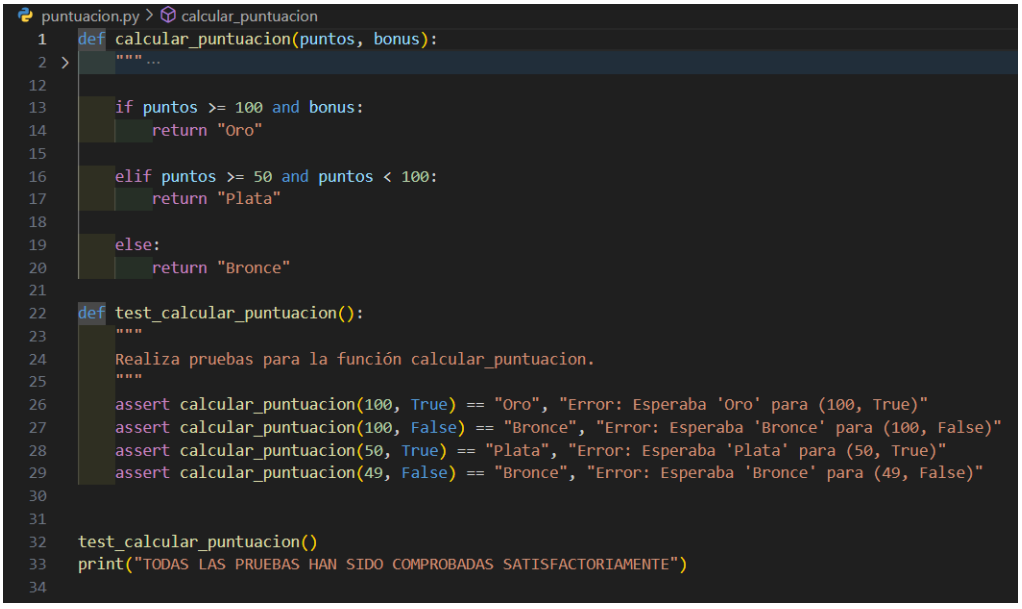
### Ejercicio email



```python
import re

def validar_email(email):
    """..."""
    # patrón de expresión regular para validar el correo electrónico

    patron = r'^[a-zA-Z0-9._]+@[a-zA-Z0-9-]+\.[a-zA-Z]{2,4}$'

    # la función match de re para verificar si el email coincide con el patrón

    if re.match(patron, email):
        return True
    else:
        return False

# Función de prueba para validar la funcionalidad de validar_email

def test_validar_email():
    """..."""
    assert validar_email("juan.perez@empresa.com") == True, "Error: Esperaba True para 'juan.perez@empresa.com'"
    assert validar_email("maria@dominio") == False, "Error: Esperaba False para 'maria@dominio'"
    assert validar_email("pedro#2024@mail.org") == False, "Error: Esperaba False para 'pedro#2024@mail.org'"

test_validar_email()
print("Todas las pruebas pasaron exitosamente.")
```

El código proporciona una función calcular_puntuacion para determinar calificaciones basadas en puntos y un estado de bonus, devolviendo "Oro", "Plata" o "Bronce" según las condiciones establecidas. Además, incluye pruebas unitarias mediante la función test_calcular_puntuacion para asegurar que la lógica funcione correctamente, verificando diferentes combinaciones de puntos y el estado del bonus.

### Ejercicio Puntuación



```python
def calcular_puntuacion(puntos, bonus):
    """..."""

    if puntos >= 100 and bonus:
        return "Oro"

    elif puntos >= 50 and puntos < 100:
        return "Plata"

    else:
        return "Bronce"

def test_calcular_puntuacion():
    """
    Realiza pruebas para la función calcular_puntuacion.
    """
    assert calcular_puntuacion(100, True) == "Oro", "Error: Esperaba 'Oro' para (100, True)"
    assert calcular_puntuacion(100, False) == "Bronce", "Error: Esperaba 'Bronce' para (100, False)"
    assert calcular_puntuacion(50, True) == "Plata", "Error: Esperaba 'Plata' para (50, True)"
    assert calcular_puntuacion(49, False) == "Bronce", "Error: Esperaba 'Bronce' para (49, False)"


test_calcular_puntuacion()
print("TODAS LAS PRUEBAS HAN SIDO COMPROBADAS SATISFACTORIAMENTE")
```

Este código también proporciona una función calcular_puntuacion que determina las calificaciones basadas en puntos y el estado del bonus, devolviendo "Oro", "Plata" o "Bronce" según las condiciones establecidas. Incluye pruebas unitarias mediante la función test_calcular_puntuacion para asegurar que la lógica funcione correctamente, verificando diferentes combinaciones de puntos y el estado del bonus.

**Ejercicio teléfonos**

telefonos.py > ...

```python
import re

def validar_telefono(numero):
    patron = r'^\+\d{2}-\d{3}-\d{3}-\d{4}$'
    return bool(re.match(patron, numero))

# Pruebas con assert
def test_validar_telefono():
    assert validar_telefono("+12-345-678-9012") == True  # Válido
    assert validar_telefono("12-345-678-9012") == False  # Falta el "+"
    assert validar_telefono("+123-456-789-0123") == False  # Prefijo de país incorrecto (debe ser 2 dígitos)
    assert validar_telefono("+12-3456-789-012") == False  # Error en los bloques de números


test_validar_telefono()
print("Pruebas satisfactoriamente aprobadas")
```

El código define una función validar_telefono que utiliza expresiones regulares para validar números de teléfono en un formato específico (+dd-ddd-ddd-dddd). Además, incluye pruebas unitarias mediante la función test_validar_telefono para verificar diferentes formatos de números de teléfono, asegurando que solo los números válidos pasen la validación.