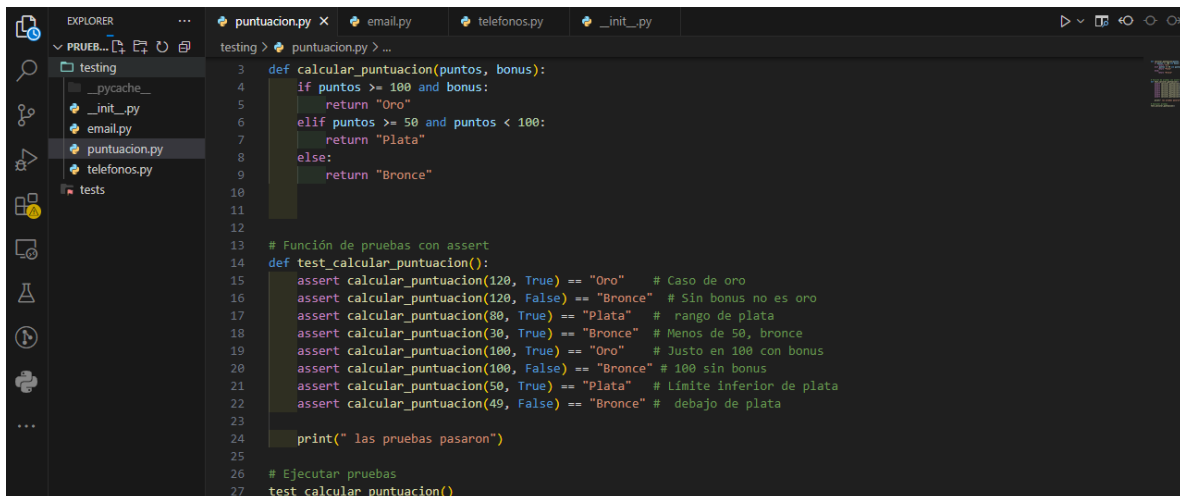


Iniciando con las pruebas

Calcular puntuación

1. Aquí La función `calcular_puntuacion` (`puntos`, `bonus`) evalúa una puntuación y devuelve una categoría según el siguiente criterio:
2. Si la puntuación es **100 o más** y el parámetro `bonus` es `True`, se devuelve **"Oro"**
3. Si la puntuación está en el rango **entre 50 y 99** (inclusive), se devuelve **"Plata"**
4. Para cualquier otro caso (menos de 50 o sin cumplir condiciones previas), se devuelve **"Bronce"**

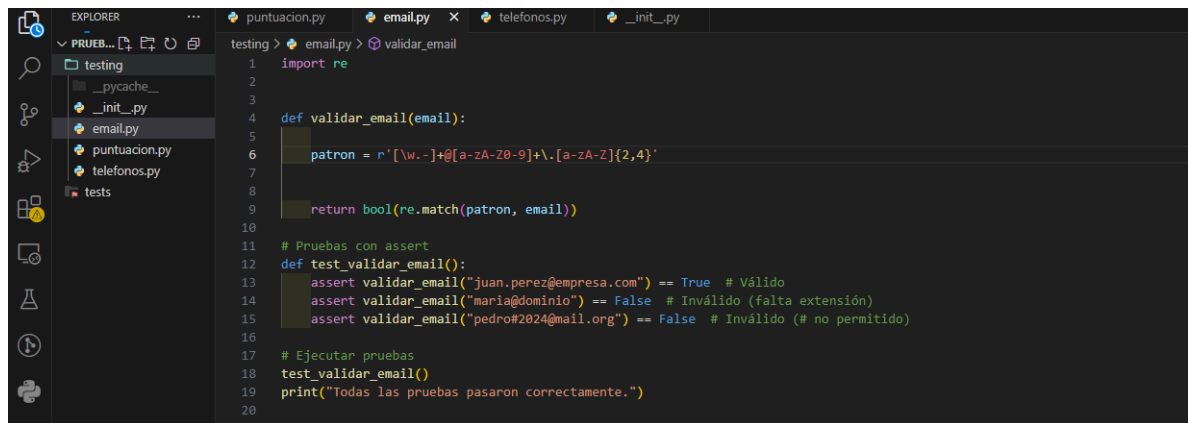
Y ponemos en práctica las pruebas con `assert calcular_puntuacion(120, True) == "Oro"` # Caso de oro.



```
3 def calcular_puntuacion(puntos, bonus):
4     if puntos >= 100 and bonus:
5         return "Oro"
6     elif puntos >= 50 and puntos < 100:
7         return "Plata"
8     else:
9         return "Bronce"
10
11
12
13 # Función de pruebas con assert
14 def test_calcular_puntuacion():
15     assert calcular_puntuacion(120, True) == "Oro" # Caso de oro
16     assert calcular_puntuacion(120, False) == "Bronce" # Sin bonus no es oro
17     assert calcular_puntuacion(80, True) == "Plata" # rango de plata
18     assert calcular_puntuacion(30, True) == "Bronce" # Menos de 50, bronce
19     assert calcular_puntuacion(100, True) == "Oro" # Justo en 100 con bonus
20     assert calcular_puntuacion(100, False) == "Bronce" # 100 sin bonus
21     assert calcular_puntuacion(50, True) == "Plata" # Límite inferior de plata
22     assert calcular_puntuacion(49, False) == "Bronce" # debajo de plata
23
24     print(" las pruebas pasaron")
25
26 # Ejecutar pruebas
27 test_calcular_puntuacion()
```

Validar email

1. La función validar_email(email) verifica si una dirección de correo electrónico es válida utilizando una expresión regular
2. Devuelve un **booleano** true si es valido y false si no lo es
3. Se usa una **expresión regular** para definir un patrón que debe cumplir el correo electrónico

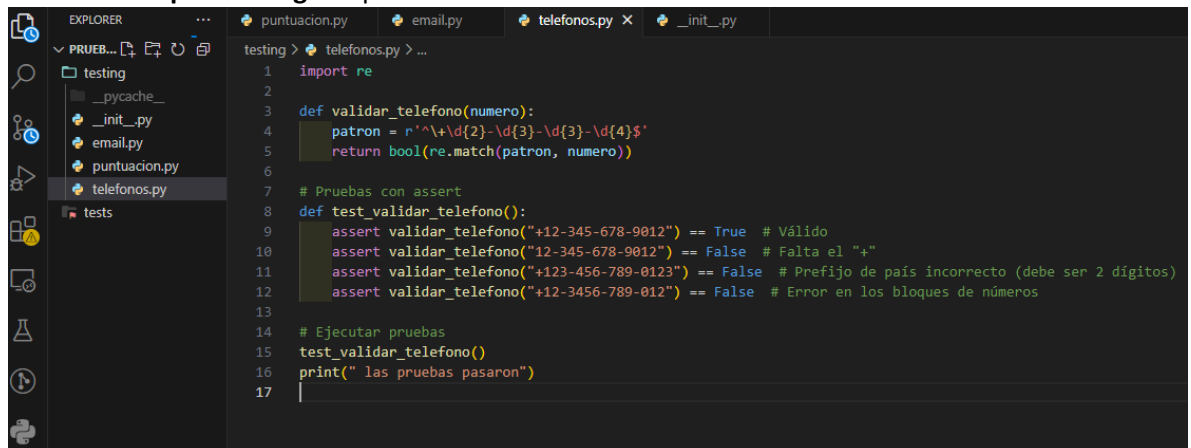


```
EXPLORER
  PRUEBAS...
    testing
      _pycache_
      _init_.py
      email.py
      puntuacion.py
      telefonos.py
      tests

testing > email.py > validar_email
1  import re
2
3
4  def validar_email(email):
5
6      patron = r'[\w.-]+@[a-zA-Z0-9]+\.[a-zA-Z]{2,4}'
7
8
9      return bool(re.match(patron, email))
10
11 # Pruebas con assert
12 def test_validar_email():
13     assert validar_email("juan.perez@empresa.com") == True # Válido
14     assert validar_email("maria@dominio") == False # Inválido (falta extensión)
15     assert validar_email("pedro#2024@mail.org") == False # Inválido (# no permitido)
16
17 # Ejecutar pruebas
18 test_validar_email()
19 print("Todas las pruebas pasaron correctamente.")
20
```

Validar teléfono

1. La función validar_telefono(numero) verifica si un número de teléfono tiene el formato correcto según el patrón
2. Devuelve un **booleano** true si es valido false si no lo es
3. Se usa una **expresión regular** para definir el formato del número de teléfono



```
EXPLORER
  PRUEBAS...
    testing
      _pycache_
      _init_.py
      email.py
      puntuacion.py
      telefonos.py
      tests

testing > telefonos.py > ...
1  import re
2
3
4  def validar_telefono(numero):
5
6      patron = r'^+\d{2}-\d{3}-\d{3}-\d{4}$'
7      return bool(re.match(patron, numero))
8
9 # Pruebas con assert
10 def test_validar_telefono():
11     assert validar_telefono("+12-345-678-9012") == True # Válido
12     assert validar_telefono("12-345-678-9012") == False # Falta el "+"
13     assert validar_telefono("+123-456-789-0123") == False # Prefijo de país incorrecto (debe ser 2 dígitos)
14     assert validar_telefono("+12-3456-789-012") == False # Error en los bloques de números
15
16 # Ejecutar pruebas
17 test_validar_telefono()
18 print(" las pruebas pasaron")
19
```

