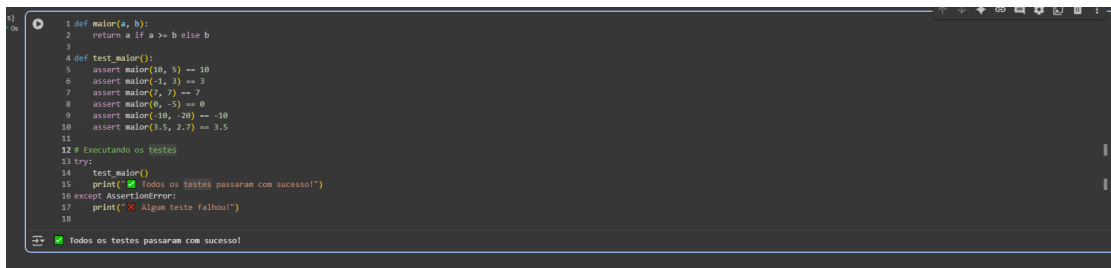


1

```
def maior(a, b):
    return a if a >= b else b

def test_maior():
    assert maior(10, 5) == 10
    assert maior(-1, 3) == 3
    assert maior(7, 7) == 7
    assert maior(0, -5) == 0
    assert maior(-10, -20) == -10
    assert maior(3.5, 2.7) == 3.5

# Executando os testes
try:
    test_maior()
    print("✅ Todos os testes passaram com sucesso!")
except AssertionError:
    print("❌ Algum teste falhou!")
```



```
1 def maior(a, b):
2     return a if a >= b else b
3
4 def test_maior():
5     assert maior(10, 5) == 10
6     assert maior(-1, 3) == 3
7     assert maior(7, 7) == 7
8     assert maior(0, -5) == 0
9     assert maior(-10, -20) == -10
10    assert maior(3.5, 2.7) == 3.5
11
12 # Executando os testes
13 try:
14     test_maior()
15     print("✅ Todos os testes passaram com sucesso!")
16 except AssertionError:
17     print("❌ Algum teste falhou!")
18
```

✅ Todos os testes passaram com sucesso!

2

```
def inverter_string(texto):
    return texto[::-1] # <-- o return devolve a string invertida

# Exemplos práticos
entradas = ["thiago", "radar", "", "Python", "67890"]

print("🔍 Testando o return da função inverter_string:\n")
for txt in entradas:
    resultado = inverter_string(txt)
    print(f"Entrada: '{txt}' --> Saída (return): '{resultado}'")
```

```
1 def inverter_string(texto):
2     return texto[::-1] # <-- o return devolve a string invertida
3
4 # Exemplos práticos
5 entradas = ["thiago", "radar", "", "Python", "67890"]
6
7 print("Testando o return da função inverter_string:\n")
8 for txt in entradas:
9     resultado = inverter_string(txt)
10    print(f"Entrada: '{txt}' --> Saída (return): '{resultado}'")
11
```

Testando o return da função inverter_string:

Entrada: 'thiago'	--> Saída (return): 'ogaiht'
Entrada: 'radar'	--> Saída (return): 'radar'
Entrada: ''	--> Saída (return): ''
Entrada: 'Python'	--> Saída (return): 'nohtyP'
Entrada: '67890'	--> Saída (return): '09876'

3

```
def valida_cpf(cpf):
    return cpf.isdigit() and len(cpf) == 11

entradas = [
    "12345678901", # válido
    "123", # muito curto
    "abc12345678", # contém letras
    "00000000000", # válido (mesmo não sendo real)
    "1234567890a", # contém letra no final
    "98765432109" # válido
]

print("Resultados da função valida_cpf:\n")
for cpf in entradas:
    print(f"Entrada: {cpf} --> Saída (return): {valida_cpf(cpf)}")
```

```

1 def valida_cpf(cpf):
2     return cpf.isdigit() and len(cpf) == 11
3
4 entradas = [
5     "12345678901", # válido
6     "123",         # muito curto
7     "abc12345678", # contém letras
8     "00000000000", # válido (mesmo não sendo real)
9     "1234567890a", # contém letra no final
10    "98765432109"  # válido
11 ]
12
13 print("🔍 Resultados da função valida_cpf:\n")
14 for cpf in entradas:
15     print(f"Entrada: {cpf} --> Saída (return): {valida_cpf(cpf)}")
16

```

➡ 🔍 Resultados da função valida_cpf:

```

Entrada: 12345678901 --> Saída (return): True
Entrada: 123 --> Saída (return): False
Entrada: abc12345678 --> Saída (return): False
Entrada: 00000000000 --> Saída (return): True
Entrada: 1234567890a --> Saída (return): False
Entrada: 98765432109 --> Saída (return): True

```