Reto 5: Funciones

1. Objetivos:

- Practicar la declaración de funciones
- Practicar la definición de parámetros y su uso dentro de la función
- Practicar el uso de los valores retornados por una función
- Practicar el uso de funciones para evitar la repetición de código

---

2. Desarrollo:

a) Función `numero_es_par`

Debajo tienes una función incompleta:

```
def numero_es_par(numero):

   return numero % 2 == 0
```

Dicha función debería de tomar un parámetro `numero`, checar si el número es par, regresar `True` si el número es par y regresar `False` si el número no es par.

Completa la función para que el código de abajo (que realiza `tests` de la función) regrese todos los `resultados esperados`.

> **Pista**: Si no conoces el operador `módulo (%)`, pídele al experto que explique su funcionamiento.

> **PD**: Si no entiendes la función test_funcion no te preocupes, por el momento sólo está ahí para probar tu código y que sepas si tu función funciona correctamente.

```
def test_funcion(funcion_a_probar, num_de_errores, contador, parametros=[], resultado_esperado=None):
    resultado_test = funcion_a_probar(*parametros)
    print(f'Test {contador}: Resultado esperado es `{resultado_esperado}`, obtuvimos `{resultado_test}`')
    if resultado_test != resultado_esperado:
        num_de_errores += 1

    return num_de_errores


print("== Tests numero_es_par==\n")

contador_de_tests = 0
errores = 0

contador_de_tests += 1
errores = test_funcion(numero_es_par, errores, contador_de_tests, parametros=[2], resultado_esperado=True)
contador_de_tests += 1
errores = test_funcion(numero_es_par, errores, contador_de_tests, parametros=[3], resultado_esperado=False)
contador_de_tests += 1
errores = test_funcion(numero_es_par, errores, contador_de_tests, parametros=[0], resultado_esperado=True)
contador_de_tests += 1
errores = test_funcion(numero_es_par, errores, contador_de_tests, parametros=[127], resultado_esperado=False)
contador_de_tests += 1
errores = test_funcion(numero_es_par, errores, contador_de_tests, parametros=[-88], resultado_esperado=True)
contador_de_tests += 1
errores = test_funcion(numero_es_par, errores, contador_de_tests, parametros=[-1349], resultado_esperado=False)

print(f'\nErrores encontrados: {errores}')
```

```
== Tests numero_es_par==

    Test 1: Resultado esperado es `True`, obtuvimos `True`
    Test 2: Resultado esperado es `False`, obtuvimos `False`
    Test 3: Resultado esperado es `True`, obtuvimos `True`
    Test 4: Resultado esperado es `False`, obtuvimos `False`
    Test 5: Resultado esperado es `True`, obtuvimos `True`
```

```
Test 6: Resultado esperado es `False`, obtuvimos `False`

Errores encontrados: 0
```

▶ Solución

b) Reutilización de código

Debajo tenemos algo de código.

```
resultado_1 = 34 * 100 / 100
print(f'34 es el {resultado_1}% del número 100\n')

resultado_2 = 57 * 100 / 127
print(f'57 es el {resultado_2}% del número 127\n')

resultado_3 = 12 * 100 / 228
print(f'12 es el {resultado_3}% del número 228\n')

resultado_4 = 87 * 100 / 90
print(f'87 es el {resultado_4}% del número 90\n')

resultado_5 = 1 * 100 / 999
print(f'1 es el {resultado_5}% del número 999\n')

resultado_6 = 66 * 100 / 66
print(f'66 es el {resultado_6}% del número 66\n')
```

```
34 es el 34.0% del número 100

57 es el 44.881889763779526% del número 127

12 es el 5.2631578947368425% del número 228

87 es el 96.66666666666667% del número 90

1 es el 0.1001001001001001% del número 999

66 es el 100.0% del número 66
```

Este código funciona correctamente, pero como puedes ver, estamos escribiendo el mismo código una y otra vez. En la celda debajo, escribe una función que realice la operación matemática que estamos realizando arriba, para que podamos reusarla múltiples veces para obtener los resultados que queremos. Llena también los `prints` de manera que podamos leer el resultado en un formato comprensible.

> Reto extra: Si quieres un reto extra, escribe también una función que genere las strings que vamos a pasar a los `prints`, para no tener que escribirlas desde 0 cada vez.

```
## Tu función va aquí
def porcentaje(muestra, total):
    return muestra * 100 / total

def mensaje(muestra, total, porcentaje):
    return f'{muestra} es el {porcentaje}% del número {total}'


resultado_1 = porcentaje(34, 100)
print(mensaje(34,100,porcentaje(34, 100)))

resultado_1 = porcentaje(57, 127)
print(mensaje(57,127,porcentaje(57, 127)))

resultado_1 = porcentaje(87, 90)
print(mensaje(87,90,porcentaje(87, 90)))

resultado_1 = porcentaje(1, 999)
print(mensaje(1,999,porcentaje(1, 999)))

resultado_1 = porcentaje(66, 66)
print(mensaje(66,66,porcentaje(66, 66)))
```

```
34 es el 34.0% del número 100
57 es el 44.881889763779526% del número 127
87 es el 96.66666666666667% del número 90
1 es el 0.1001001001001001% del número 999
66 es el 100.0% del número 66
```

▶ Solución

c) Función `acceso_autorizado`

Debajo tenemos un conjunto de datos que tiene información de varios usuarios de una plataforma web. Este diccionario relaciona `usernames` con un rol y (a veces) con un nip de acceso:

```
usuarios = {
    "manolito_garcia": {
        "rol": "admin"
    },
    "sebas_macaco_23": {
        "rol": "editor",
        "nip_de_acceso": 3594
    },
    "la_susanita_maestra": {
        "rol": "admin"
    },
    "pepe_le_pu_88": {
        "rol": "lector"
    },
    "jonny_bravo_estuvo_aqui": {
        "rol": "editor",
        "nip_de_acceso": 9730
    },
    "alfonso_torres_69": {
        "rol": "editor",
        "nip_de_acceso": 2849
    },
    "jocosita_99": {
        "rol": "lector"
    }
}
```

Nuestra plataforma tiene 3 roles:

1. Admin: estos pueden editar información sin necesitar un nip de acceso.
2. Editor: pueden editar información sólo si escriben correctamenta su nip de acceso.
3. Lector: no pueden editar, sólo ver la información, no necesitan nip de acceso.

En la celda debajo, crea una función llamada `nivel_de_acceso_para_username` que reciba 3 parámetros:

1. `base_de_datos`: que será siempre nuestro diccionario `usuarios`.
2. `username`: El `username` del usuario que está solicitando acceso.
3. `nip_de_acceso`: El nip de acceso, que puede ser `None` en el caso de que el usuario no tenga uno (o que haya olvidado escribirlo a la hora de pedir acceso.

Con estos 3 parámetros, nuestra función tiene que regresar uno de los códigos de acceso siguientes:

- `0`: Acceso denegado (esto sucede si el rol es `editor` y el `nip_de_acceso` es incorrecto).
- `1`: Modo edición autorizada (esto sucede si el rol es `admin` o si el rol es `editor` y el `nip_de_acesso` es correcto).
- `2`: Modo lectura autorizada (esto sucede si el rol es `lector`).

Después, corre los tests para asegurarte de que tu función es correcta.

> Tip: Recuerda que puedes "anidar" `sentencias if` dentro de otras `sentencias if`.

> Reto extra: Agrega un chequeo que regrese `0` si el `username` que recibió tu función no existe en la base de datos.

```
## Tu función va aquí
def nivel_de_acceso_para_username(base_de_datos, username,  nip_de_acceso):
    if username not in base_de_datos:
        return 0
```

```
    if base_de_datos[username]["rol"] == "admin":
        return 1
    elif base_de_datos[username]["rol"] == "lector":
        return 2
    else:
        if base_de_datos[username]["nip_de_acceso"] == nip_de_acceso:
            return 1
        else:
            return 0


print("== Tests nivel_de_acceso_para_username==\n")

contador_de_tests = 0
errores = 0

contador_de_tests += 1
errores = test_funcion(nivel_de_acceso_para_username, errores, contador_de_tests,
                       parametros=[usuarios, "manolito_garcia", None], resultado_esperado=1)
contador_de_tests += 1
errores = test_funcion(nivel_de_acceso_para_username, errores, contador_de_tests,
                       parametros=[usuarios, "sebas_macaco_23", 3594], resultado_esperado=1)
contador_de_tests += 1
errores = test_funcion(nivel_de_acceso_para_username, errores, contador_de_tests,
                       parametros=[usuarios, "jonny_bravo_estuvo_aqui", 9999], resultado_esperado=0)
contador_de_tests += 1
errores = test_funcion(nivel_de_acceso_para_username, errores, contador_de_tests,
                       parametros=[usuarios, "pepe_le_pu_88", None], resultado_esperado=2)
contador_de_tests += 1
errores = test_funcion(nivel_de_acceso_para_username, errores, contador_de_tests,
                       parametros=[usuarios, "alfonso_torres_69", None], resultado_esperado=0)

# Corre el siguiente código sólo si decidiste realizar el reto extra
#
# contador_de_tests += 1
# errores = test_funcion(nivel_de_acceso_para_username, errores, contador_de_tests,
#                        parametros=[usuarios, "los_yeah_yeahs_97", 1345], resultado_esperado=0)

print(f'\nErrores encontrados: {errores}')
```

```
== Tests nivel_de_acceso_para_username==

    Test 1: Resultado esperado es `1`, obtuvimos `1`
    Test 2: Resultado esperado es `1`, obtuvimos `1`
    Test 3: Resultado esperado es `0`, obtuvimos `0`
    Test 4: Resultado esperado es `2`, obtuvimos `2`
    Test 5: Resultado esperado es `0`, obtuvimos `0`

    Errores encontrados: 0
```

▶ Solución