# Talle #2 Pruebas de Software

## 1-Validación de Nombre de Usuario.

Especificaciones del Sistema

La función validar_usuario (nombre) debe cumplir:

1. Longitud entre 5 y 15 caracteres.
2. Sin espacios.
3. Solo letras (mayúsculas/minúsculas) y números.

Paso 1: crear una tabla con 10 casos de prueba usando las técnicas aprendidas.

| Caso | Entrada | Resultado Esperado | Técnica Aplica | Observaciones |
|---|---|---|---|---|
| 1 | “Ana” | False | BVA (Límite Inferior) | Muy Corto. |
| 2 | “12345” | True | Partición valida. | Valido porque tiene min 5 caracteres de números. |
| 3 | “Juan_Pérez” | False | Carácter especial (_) | Solo puede tener núm. y letras |
| 4 | “A123456789012345” | False | BVA (límite superior) | Tiene más de 15 caracteres. |
| 5 | “ABBCDE” | True | Partición valida | Valido porque tiene min 5 caracteres de letras. |
| 6 | “alb2c3” | True | Combinación de letras y núm. | Valido porque combina letras y números |
| 7 | “1234567890123456” | False | BVA (límite superior) | Tiene más de 15 caracteres. |
| 8 | “NombreUsuario!” | False | Carácter especial (!) | Solo puede tener núm. y letras |
| 9 | “UsuarioValido12” | True | Partición valida | |
| 10 | “Nombre Usuario” | False | Contiene espacio | No puede contener espacios |

Paso 2: Implementar pruebas en Python usando assert.

**Creamos la función validar_usuario:**

```
def validar_usuario(nombre):
    """
    Esta funcion verifica si un nombre de usuario es valido segun los siguientes instrucciones:
```

```
    - La longitud del nombre debe estar entre 5 y 15 caracteres.
    - No debe contener espacios.
    - Solo debe contener letras (mayúsculas/minúsculas) y números.

    Parametro:
    nombre (str): Para validar el nombre de usuario.

    Retorno:
    bool: True si el nombre de usuario es válido, False si no lo es.
    """
    if len(nombre) < 5 or len(nombre) > 15:
        """
        Verifica si la longitud del nombre de usuario es menor que 5 o mayor que 15 caracteres.
        Si es así, el nombre de usuario no es válido.
        """
        return False
    if " " in nombre:
        """
        Verifica si el nombre de usuario contiene espacios.
        Si contiene espacios, el nombre de usuario no es válido.
        """
        return False
    for c in nombre:
        """
        Recorre cada carácter en el nombre de usuario para verificar si es alfanumérico.
        Si encuentra algún carácter que no sea alfanumérico, el nombre de usuario no es válido.
        """
        if not c.isalnum():
            return False
        """
        Si todas las verificaciones anteriores pasan, el nombre de usuario es válido.
        """
    return True
```

**Creamos las pruebas para validar_usuario:**

```
from inicio import paso2

def test_validar_usuario():
    """
    Esta funcion realiza varias pruebas para verificar que la func validar_usuario(nombre) funcione correctamente.

    Las pruebas incluyen casos validos ey casos invalidos:
    - Casos False:
        1. Nombre muy corto.
        2. Caracter especial (_) no permitido.
        3. Convinacion de numeros y letras con mas de 15 caracteres.
```

```
    4. Usuario numerico con mas de 15 caracteres.
    5. Caracter especial (!) no permitido.
    6. Contiene espacio no permitido.
"""
#casos False
assert paso2.validar_usuario("Ana") == False
assert paso2.validar_usuario("Juan_Perez") == False
assert paso2.validar_usuario("ABC23456789012345") == False
assert paso2.validar_usuario("1234567890123456") == False
assert paso2.validar_usuario("UsuarioNombre!") == False
assert paso2.validar_usuario("Usuario Nombre") == False

"""
- Casos True:
    1. Nombre de usuario numerico con minimo 5 caracteres.
    2. Nombre de usuario con letras con minimo 5 caracteres.
    3. Nombre de usuario convinando letras miniscula y numeros de 6 caracteres.
    4. Nombre de usuario con letras y numeros no excede los 15 caracteres.
"""
#casos True
assert paso2.validar_usuario("12345") == True
assert paso2.validar_usuario("ABCDE") == True
assert paso2.validar_usuario("a1b2c3") == True
assert paso2.validar_usuario("UsuarioValido12") == True

"""
Estas pruebas de realizaron siguiendo las siguientes instrucciones:
    - La longitud del nombre debe estar entre 5 y 15 caracteres.
    - No debe contener espacios.
    - Solo debe contener letras (mayúsculas/minúsculas) y números.
"""
```

**¿Por qué fallaron ciertos casos?**

> Lo adjunte en las observaciones del cuadro porque fueron validos y el porque fallaron cada uno de los casos.

**¿Cómo mejorar los casos de prueba para cubrir más escenarios?**

- Incluir más casos de prueba en los límites de longitud (e.g., exactamente 5 y exactamente 15 caracteres).
- Probar con diferentes combinaciones de letras y números.
- Asegurar que la función maneje correctamente caracteres válidos y detecte caracteres inválidos de manera consistente.
- Probar nombres de usuario que solo contengan letras o solo números.

# 2-Sistema de Puntuación de Juegos.

Especificaciones

La función calcular_puntuacion (puntos, bonus) retorna:

1. "Oro" si puntos >= 100 y bonus == True.
2. "Plata" si puntos >= 50 y puntos < 100.
3. "Bronce" si puntos < 50 o bonus == False.

Paso 1: Crear una tabla que combine todas las condiciones:

| Puntos | Bonus | Resultado |
|---|---|---|
| 100 | True | Oro |
| 120 | True | Oro |
| 50 | True | Plata |
| 80 | True | Plata |
| 75 | True | Plata |
| 15 | False | Bronce |
| 20 | False | Bronce |
| 30 | False | Bronce |
| 2 | True | Bronce |
| 25 | False | Bronce |

Paso 2: Implementar pruebas:

**Creamos las pruebas para calcular_puntuacion:**

```
def test_calcular_puntuacion ():
    """
    Esta función realiza varias pruebas para verificar que la func calcular_puntuacion (puntos, bonus)
    funcione correctamente.

    Las pruebas incluyen:
    ---> Casos donde se debe retornar "Oro".
    ---> Casos donde se debe retornar "Plata".
    ---> Casos donde se debe retornar "Bronce".
    """
    # Casos Oro
    assert ejercicio2.calcular_puntuacion (100, True) == "Oro"
    assert ejercicio2.calcular_puntuacion (120, True) == "Oro"

    # Casos Plata
```

```
    assert ejercicio2.calcular_puntuacion (50, True) == "Plata"
    assert ejercicio2.calcular_puntuacion (80, True) == "Plata"
    assert ejercicio2.calcular_puntuacion (75, True) == "Plata"

    # Casos Bronce
    assert ejercicio2.calcular_puntuacion (15, False) == "Bronce"
    assert ejercicio2.calcular_puntuacion (20, False) == "Bronce"
    assert ejercicio2.calcular_puntuacion (30, False) == "Bronce"
    assert ejercicio2.calcular_puntuacion (2, True) == "Bronce"
    assert ejercicio2.calcular_puntuacion (25, False) == "Bronce"
```

**Creamos las pruebas para calcular_puntuacion ():**

```
def calcular_puntuacion (puntos, bonus):
    """
    Esta función calcula la puntuación del juego según las siguientes instrucciones:
    ---> Oro si puntos >= 100 y bonus == True.
    ---> Plata si puntos >= 50 y puntos < 100.
    ---> Bronce si puntos < 50 o bonus == False.

    Parámetros:
    puntos == int: La cantidad de puntos obtenidos.
    bonus == bool: Indica si se ha recibido un bonus.

    Retorno:
    str: "Oro", "Plata" o "Bronce" según la puntuación.
    """
    if puntos >= 100 and bonus:
        return "Oro"
    elif puntos >= 50 and puntos < 100 and bonus:
        return "Plata"
    else: puntos < 50
    return "Bronce"
```

# 3-Validación de Email.

Especificaciones:

1. La función validar_email(email) debe cumplir:
2. Formato usuario@dominio.extension.
3. usuario: Solo letras, números, ., _, y ``.
4. dominio: Letras, números, y guiones.
5. extensión: 2 a 4 letras.

**Creamos la función para Validar el Email:** Importamos el modulo “re” que pertenece a la biblioteca estándar de Python:

```
import re

def validar_email(email):
```

```
    """
    Esta función valida un email según las siguientes instrucciones:
    ---> Formato usuario@dominio.extension.
    ---> usuario: Solo letras, números, ., _, y `.
    ---> dominio: Letras, números, y guiones.
    ---> extensión: 2 a 4 letras.

    Paramito:
    email "str": para validar el correo.

    Returns:
    bool: True si el email es valido, False si no lo es.

    re: pertenece a la biblioteca estándar de python e incluye el patrón utilizado.
    """
    patron = r'^[a-zA-Z0-9._`]+@[a-zA-Z0-9-]+\.[a-zA-Z]{2,4}$'
    return re.match(patron, email) is not None
```

**Creamos las pruebas para Validar email:**

```
def test_validar_email():
    """
    Esta función realiza pruebas para verificar que la funcion validar_email(email)
    se ejecute correctamente.

    Las pruebas incluyen:
    ---> Casos True:
        Correos con todas sus características: usuario@dominio.extencion e incluso un - en
el dominio.

    ---> Casos False:
        1.Correo con falta de extensión
        2.Caracter # no permitido
        3.Extencion incorrecta
        4.Doble punto en el dominio
        5.Sin extensión
        6.Extencion incompleta
        7.Sin usuario
        8.Sin dominio

    """
    # Casos True
    assert ejercicio3.validar_email("danielpacsota93@gmail.com") == True
    assert ejercicio3.validar_email("usuario@dominio-.com") == True

    # Casos False
    assert ejercicio3.validar_email("maria@dominio") == False
    assert ejercicio3.validar_email("pedro#2024@mail.org") == False
    assert ejercicio3.validar_email("user@dominio.extended") == False
    assert ejercicio3.validar_email("user@domain..com") == False
```

```
    assert ejercicio3.validar_email("user@domain") == False
    assert ejercicio3.validar_email("usuario@dominio.c") == False
    assert ejercicio3.validar_email("@dominio.com") == False
    assert ejercicio3.validar_email("usuario@.com") == False
```

# 4-Validación de teléfono.

Ejercicio: Diseñar casos de prueba para una función que valide números de teléfono con formato +XX-XXX-XXX-XXXX.

**Creamos la función validar_telefono: i**mportamos el módulo “re” que pertenece a la biblioteca estándar de Python:

```
import re

def validar_telefono(telefono):
    """
    Esta función valida números de teléfonos que tenga el formato solicitado +XX-XXX-XXX-XXXX.

    Parámetro:
    teléfono string: Numero de teléfono a verificar.

    Retorno:
    bool: True si el número de teléfono es válido, False si no lo es.
    """
    patrón = r'^\+\d{2}-\d{3}-\d{3}-\d{4}$'
    return re.match(patron, teléfono) is not None
```

**Creamos las pruebas para Validar teléfono:**

```
def test_validar_telefono():
    """
    Esta función realiza pruebas para verificar que la función validar_telefono()
    se ejecute correctamente.

    Las pruebas incluyen:
    ---> Casos True:
        Un numero que cumple con el formato solicitado.

    ---> Casos False:
        1.Falta el signo +
        2.Codigo del país con más de 2 números
        3.Codigo de área con más de 3 números
        4.Primer segmento con más de 3 números
        5.Segundo segmento con más de 4 números
        6.Codigo de área con menos de 3 números
        7.Primer segmento con menos de 3 números
        8.Segundo segmento con menos de 4 números
```

9.Letras el código del país

"""
# Caso True
assert ejercicio4.validar_telefono("+12-345-678-9012") == True

# Casos False
assert ejercicio4.validar_telefono("12-345-678-9012") == False
assert ejercicio4.validar_telefono("+123-456-789-0123") == False
assert ejercicio4.validar_telefono("+12-3456-789-0123") == False
assert ejercicio4.validar_telefono("+12-345-6789-0123") == False
assert ejercicio4.validar_telefono("+12-345-678-01234") == False
assert ejercicio4.validar_telefono("+12-34-678-9012") == False
assert ejercicio4.validar_telefono("+12-345-67-9012") == False
assert ejercicio4.validar_telefono("+12-345-678-912") == False

**Por ultimo llamamos a todas las funciones creadas para ejecutarlas:**

from tests.tests_inicio import test_validar_usuario, test_calcular_puntuacion, test_validar_email, test_validar_telefono
"""
Despues de importar la libreria tests para el documento que creamos tests_inicio importamos las funciones que queremos finalmente llamar para ejecutar.
"""
test_validar_usuario()

test_calcular_puntuacion ()

test_validar_email()

test_validar_telefono()

.