Funciones

1. Crear una función que le solicite al usuario el ingreso de un número entero y lo retorne.
2. Crear una función que le solicite al usuario el ingreso de un número flotante y lo retorne.
3. Crear una función que le solicite al usuario el ingreso de una cadena y la retorne.
4. Escribir una función que calcule el área de un rectángulo. La función recibe la base y la altura y retorna el área.
5. Escribe una función que calcule el área de un círculo. La función debe recibir el radio como parámetro y devolver el área.
6. Crea una función que verifique si un número dado es par o impar. La función debe imprimir un mensaje indicando si el número es par o impar.
7. Crea una función que verifique si un número dado es par o impar. La función retorna True si el número es par, False en caso contrario.
8. Define una función que encuentre el máximo de tres números. La función debe aceptar tres argumentos y devolver el número más grande.
9. Diseña una función que calcule la potencia de un número. La función debe recibir la base y el exponente como argumentos y devolver el resultado.
10. Crear una función que reciba un número y retorne True si el número es primo, False en caso contrario.
11. Crear una función que (utilizando el algoritmo del ejercicio de la guia de for), muestre todos los números primos comprendidos entre entre la unidad y un número ingresado como parámetro. La función retorna la cantidad de números primos encontrados. Modularizar todo lo posible.
12. Crear una función que imprima la tabla de multiplicar de un número recibido como parámetro. La función debe aceptar parámetros opcionales (inicio y fin) para definir el rango de multiplicación. Por defecto es del 1 al 10.
13. Especializar las funciones del punto 1, 2 y 3 para hacerlas reutilizables. Agregar validaciones.

Módulos

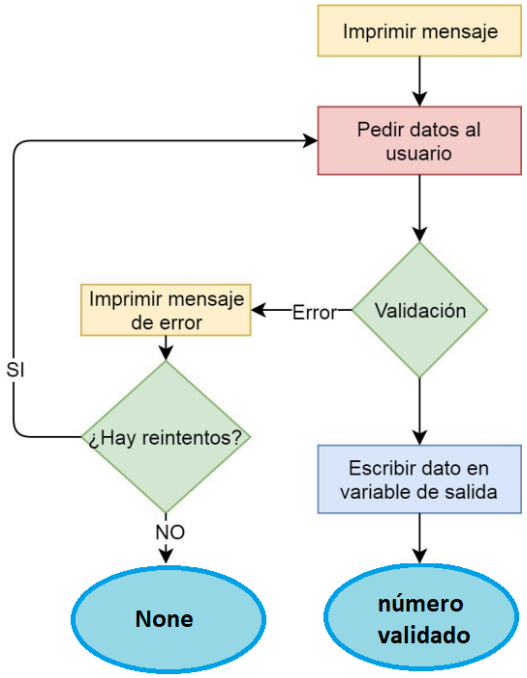
1. Realizar una función para pedir un número por consola. La misma deberá seguir el siguiente prototipo:

```
def get_int(mensaje:str, mensaje_error: str, minimo: int, maximo:int, reintentos: int) -> int|None:
    pass
```

En donde:

- **mensaje**: es el mensaje que se va a imprimir antes de pedirle al usuario el dato por consola.
- **mensaje_error**: mensaje de error en el caso de que el dato ingresado sea invalido.
- **mínimo**: valor mínimo admitido (inclusive)
- **máximo**: valor máximo admitido (inclusive)
- **reintentos**: cantidad de veces que se volverá a pedir el dato en caso de error.

En caso de que la función no haya podido conseguir un número válido, la misma retorna None.



Repetir el mismo procedimiento para un dato de tipo float.

2. Teniendo en cuenta la función del punto 1, crear la función get_string. La misma validará la longitud de la cadena ingresada dado el parámetro recibido. **El siguiente prototipo es la base para realizar el ejercicio (se puede extender):**

```
def get_string(longitud: int) -> str|None:
    pass
```

Nota: utilizar la función len.

3. Realizar los siguientes módulos:

- ***Input.py***
  - get_int()
  - get_float()
  - get_string()
- ***Validate.py***
  - validate_number()
  - validate_length()

  - ***Nota: estas funciones las tendrán que desarrollar en el módulo Validate y utilizar en el módulo Input.py para realizar las validaciones necesarias.***

Crear un repositorio en github con su apellido y nombre junto con el texto funciones_input:
Por ejemplo: **scarafilo_german_funciones_input**
Dicho repositorio deberá ser privado. Agregar a sus profesores como colaboradores.

Funciones recursivas

1. Realizar una función recursiva que calcule la suma de los primeros números naturales:

```python
def sumar_naturales(numero: int)->int:
    pass
```

2. Realizar una función recursiva que calcule la potencia de un número:

```python
def calcular_potencia(base: int, exponente: int)->int:
    pass
```

3. Realizar una función recursiva que permita realizar la suma de los dígitos de un número:

```python
def sumar_digitos(numero: int)->int:
    pass
```

4. Realizar una función para calcular el número de Fibonacci de un número ingresado por consola. La función deberá seguir el siguiente prototipo:

Definición:
La sucesión de Fibonacci comienza con los números 0 y 1, y cada número subsecuente es la suma de los dos anteriores:

- $F(0) = 0$
- $F(1) = 1$
- $F(2) = F(1) + F(0) = 1 + 0 = 1$
- $F(3) = F(2) + F(1) = 1 + 1 = 2$
- $F(4) = F(3) + F(2) = 2 + 1 = 3$
- $F(5) = F(4) + F(3) = 3 + 2 = 5$
- $F(6) = F(5) + F(4) = 5 + 3 = 8$
- $F(7) = F(6) + F(5) = 8 + 5 = 13$
- $F(8) = F(7) + F(6) = 13 + 8 = 21$
- $F(9) = F(8) + F(7) = 21 + 13 = 34$

```python
def calcular_fibonacci(numero: int)->int:
    pass
```

Nota general: en cada ejercicio al ingresar un número, se tendrá que utilizar la función **get_int** del módulo Input

Piedra, Papel o Tijera - Desafío de Programación

El clásico juego de la infancia, donde dos jugadores eligen entre tres elementos y la victoria se determina según las siguientes reglas:

- **Piedra** aplasta **Tijera** → 🏆 **Gana la Piedra**
- **Tijera** corta **Papel** → 🏆 **Gana la Tijera**
- **Papel** envuelve **Piedra** → 🏆 **Gana el Papel**
- Si ambos jugadores eligen el mismo elemento, la ronda termina en **empate**.

📌 Reglas del Juego

- La partida se juega al **mejor de 3 rondas**.
- Si un jugador (humano o máquina) logra **dos aciertos seguidos**, la partida finaliza automáticamente.
- En caso de **empate en las 3 rondas**, el juego continuará hasta que haya un ganador.

📌 Funciones a desarrollar

1️⃣ `verificar_ganador_ronda(jugador, maquina) → str`

📌 **Objetivo:** Determina quién ganó la ronda según las elecciones del jugador y la máquina.

🔹 **Parámetros:**

- `jugador` (int): Elección del jugador (1 para Piedra, 2 para Papel, 3 para Tijera).
- `maquina` (int): Elección de la máquina (1 para Piedra, 2 para Papel, 3 para Tijera).

🔹 **Retorno:**

- `"Jugador"` → Si el jugador gana la ronda.
- `"Máquina"` → Si la máquina gana la ronda.
- `"Empate"` → Si ambos eligen el mismo elemento.

2️⃣ `verificar_estado_partida(aciertos_jugador, aciertos_maquina, ronda_actual) → bool`

📌 **Objetivo:** Determina si la partida sigue en curso o ya ha finalizado.

🔹 **Parámetros:**

- `aciertos_jugador` (int): Cantidad de rondas ganadas por el jugador.
- `aciertos_maquina` (int): Cantidad de rondas ganadas por la máquina.
- `ronda_actual` (int): Número de la ronda actual.

🔹 **Retorno:**

- `True` → Si la partida sigue en curso.
- `False` → Si la partida ha finalizado (porque alguien ganó dos veces seguidas o se jugaron todas las rondas).

3️⃣ `verificar_ganador_partida(aciertos_jugador, aciertos_maquina) → str`

📌 **Objetivo:** Determina quién ganó la partida comparando los aciertos finales.

🔹 **Parámetros:**

- `aciertos_jugador` (int): Cantidad de rondas ganadas por el jugador.
- `aciertos_maquina` (int): Cantidad de rondas ganadas por la máquina.

🔹 **Retorno:**

- `"Jugador"` → Si el jugador tiene más aciertos.
- `"Máquina"` → Si la máquina tiene más aciertos.

4️⃣ `mostrar_elemento(eleccion) → str`

📌 **Objetivo:** Convierte la elección numérica en un texto legible.

🔹 **Parámetros:**

- `eleccion` (int): Número de elección (1 para Piedra, 2 para Papel, 3 para Tijera).

🔹 **Retorno:**

- `"Piedra"` cuando su `elección == 1`.
- `"Papel"` cuando su `elección == 2`.
- `"Tijera"` cuando su `elección == 3`.

5️⃣ `jugar_piedra_papel_tijera() → str`

📌 **Objetivo:** Gestiona toda la lógica del juego, usando las funciones anteriores.

🔹 **Flujo de la función:**

1. Inicia una partida donde el jugador compite contra la máquina.
2. En cada ronda, el jugador elige una opción y la máquina genera una elección aleatoria.
3. Se determina el ganador de la ronda.
4. Se verifica si la partida continúa o si alguien ha ganado.
5. Al finalizar, la función devuelve quién ganó la partida (`"Jugador"` o `"Máquina"`).

📌 Requisitos del Código

✅ Todas las funciones deben estar correctamente modularizadas.
✅ Se debe validar que el jugador solo ingrese valores válidos (1, 2 o 3).
✅ Se deben manejar posibles errores de entrada de datos.
✅ Se recomienda usar `random.randint(1,3)` para la elección de la máquina.
✅ Mostrar mensajes claros en cada ronda indicando los elementos elegidos y el estado de la partida.
✅ Crear tantos módulos como considere necesario y reutilizar los propios.