**Guía de Prácticas de Laboratorio: Python Modulo 2**

| **Programa de Formación:** | Tecnólogo en análisis y desarrollo de Software |
| --- | --- |
| **Tematica:** | Python Modulo 2: Lógica y Control de Flujo 🧠 |

**Criterios Esenciales de Entrega checklist**

**Para cada ejercicio, asegúrate de cumplir con los siguientes puntos:**

* **Git**: Usa un solo repositorio de Git, hacer uso de gitignore y hacer un commit descriptivo por cada ejercicio resuelto.
* **Entorno Virtual**: Gestiona el proyecto con uv (debe existir un archivo pyproject.toml).
* **Pruebas Unitarias**: Crea un archivo de pruebas para cada ejercicio (test\_\*.py) y verifica la funcionalidad de tu código con pytest.
* **Documentación**: Todas las funciones deben tener docstrings explicando su propósito.
* **Validacion de Tipado de Datos**: Utiliza Validaciones en las firmas de las funciones (parámetros y valor de retorno).
* **Estilo de Código**: El código debe ser legible y seguir las guías de estilo de PEP 8.

**Bloque 1: Lógica Condicional Avanzada y Bucles (Ejercicios 1-5)**

Estos ejercicios refuerzan la toma de decisiones complejas y el uso de bucles para validación y repetición controlada.

**Ejercicio 1: Sistema de Precios de Entradas de Cine**

Crea un programa que calcule el precio de una entrada de cine basándose en la edad del cliente y si es estudiante:

**Reglas:**

* Niños (menores de 12 años): $10.000.
* Jóvenes (12 a 17 años): $15.000.
* Adultos (18 años en adelante): $20.000.
* Si el cliente es estudiante (independientemente de la edad), tiene un 10% de descuento.

El programa debe pedir la edad y si es estudiante ('si' o 'no').

**Conceptos aplicados:** if/elif/else anidados, operadores lógicos (and, or), input(), int(), lower(), f-strings.

**Ejercicio 2: Intérprete de Comandos Sencillo**

Desarrolla un programa que simule un menú de consola usando la estructura match-case. El programa mostrará una lista de comandos disponibles ("guardar", "cargar", "salir") y el usuario ingresará uno

* El programa debe ejecutar una acción simulada para cada comando (ej. imprimir "Guardando archivo...").
* Si el comando no es válido, debe mostrar un mensaje de error.
* El programa debe seguir pidiendo comandos hasta que el usuario escriba "salir".
* **Conceptos aplicados**: Bucle while, match-case, input(), lower().

**Ejercicio 3: Validador de Contraseñas**

Escribe un programa que pida al usuario crear una contraseña y la valide usando un bucle while. El bucle solo terminará cuando la contraseña cumpla todos los criterios:

* Mínimo 8 caracteres de longitud.
* Contiene al menos una letra mayúscula.
* Contiene al menos un número.
* En cada intento fallido, el programa debe indicar qué regla no se cumplió.

**Conceptos aplicados**: Bucle while True, if/elif/else, len(), métodos de string (isupper(), islower(), isdigit()), break.

**Ejercicio 4: Juego de Piedra, Papel o Tijeras**

Implementa el clásico juego para jugar contra la computadora.

* El usuario elige una opción y la computadora elige una al azar.
* El programa determina el ganador basándose en las reglas (piedra vence a tijeras, tijeras a papel, papel a piedra).
* Se debe llevar un conteo de las victorias del jugador y de la computadora. El juego termina cuando uno de los dos llegue a 3 victorias.

**Conceptos aplicados**: random.choice(), bucle while, if/elif/else, contadores, f-strings.

**Ejercicio 5: Clasificador de Números (Par/Impar con Ternario)**

Crea un programa que pida un número y, usando un operador ternario, asigne a una variable el texto "Par" o "Impar". Luego, imprime el resultado. Adicionalmente, si el número es múltiplo de 5, debe imprimir un mensaje extra.

**Conceptos aplicados:** Operador ternario, operador módulo (%), if.

**Bloque 2: Manipulación Iterativa y Herramientas Avanzadas (Ejercicios 6-10)**

Aquí se aplican las herramientas avanzadas de iteración para escribir código más "Pythónico", conciso y legible.

**Ejercicio 6: Analizador de Posiciones de Letras con enumerate**

Crea una función que reciba una frase y una letra. La función debe devolver una lista con todos los índices (posiciones) en los que aparece esa letra en la frase.

* **Ejemplo**: encontrar\_indices("Hola SENA", "a") debería devolver [3, 8].

**Conceptos aplicados:** Funciones, enumerate(), bucle for, list.append().

**Ejercicio 7: Combinador de Listas con zip**

Dadas dos listas, una con nombres de estudiantes y otra con sus respectivas notas finales, crea una función que las combine para generar un diccionario. Las claves serán los nombres y los valores las notas:

* Luego, itera sobre el diccionario resultante para imprimir un reporte del tipo: "El estudiante [Nombre] tiene una nota de [Nota]".

**Conceptos aplicados:** Funciones, zip(), dict(), bucle for sobre diccionarios.

**Ejercicio 8: Filtrado de Datos con List Comprehensions**

Dada una lista de números [-5, 10, -15, 20, -25, 30], utiliza una list comprehension para crear tres nuevas listas:

* Una lista con solo los números positivos.
* Una lista con los cuadrados de todos los números.
* Una lista de strings que diga "positivo" o "negativo" para cada número, usando un ternario dentro de la comprensión.
* **Conceptos aplicados:** List comprehensions, operador ternario.

**Ejercicio 9: Transformación de Datos con Dictionary Comprehensions**

Tienes una lista de productos, donde cada producto es un diccionario: [{"nombre": "Camisa", "precio": 50000}, {"nombre": "Pantalón", "precio": 80000}]:

* Usa una dictionary comprehension para crear un nuevo diccionario donde los nombres de los productos sean las claves y los precios con un 19% de IVA incluido sean los valores.

**Conceptos aplicados**: Dictionary comprehensions, acceso a valores de diccionario.

**Ejercicio 10: Transposición de una Matriz**

Crea una función que reciba una matriz (lista de listas) y devuelva su transpuesta. La transpuesta se logra intercambiando filas por columnas.

* Ejemplo: [[1, 2, 3], [4, 5, 6]] se convierte en [[1, 4], [2, 5], [3, 6]].
* Resuelve este problema usando bucles for anidados y luego intenta resolverlo con una list comprehension anidada.

**Conceptos aplicados**: Listas anidadas (matrices), bucles anidados, list comprehensions anidadas.

**Bloque 3: Algoritmos y Proyectos de Lógica Aplicada (Ejercicios 11-15)**

Estos ejercicios integran todos los conceptos del módulo en la solución de problemas algorítmicos más complejos.

**Ejercicio 11: Validador de Cédula (Algoritmo Simple)**

Escribe una función que valide un número de cédula (como string) basado en una regla simple: la suma de sus dígitos debe ser un número par. La función debe devolver True si es válido y False si no:

* Adicionalmente, el programa principal debe pedir al usuario su cédula hasta que ingrese una válida, usando un bucle.

**Conceptos aplicados:** Funciones, bucle while, conversión de tipos (str a int), bucle for sobre un string.

**Ejercicio 12: Simulador de Lanzamiento de Dados**

* Crea un programa que simule el lanzamiento de dos dados 10,000 veces.
* Usa un diccionario para contar la frecuencia de cada posible suma de los dados (de 2 a 12).
* Al final, imprime un reporte con la frecuencia de cada suma.

**Conceptos aplicados:** random.randint(), bucles, diccionarios como contadores, método get().

**Ejercicio 13: Aventura de Texto Simple**

Diseña un pequeño juego de aventura basado en texto. El jugador comienza en una habitación y puede tomar decisiones ('ir al norte', 'abrir cofre'):

* El juego debe tener al menos 3 habitaciones y un estado final (ganar o perder).
* Utiliza un bucle while para el bucle principal del juego y estructuras if/elif/else para manejar las decisiones del jugador y el estado del juego.

**Conceptos integrados:** Bucles, condicionales, manejo de estado con variables, input().

**Ejercicio 14: Juego del Ahorcado (Hangman)**

Diseña una versión para consola del clásico juego del "Ahorcado". El programa debe ser capaz de gestionar toda la lógica del juego, desde la selección de la palabra hasta determinar si el jugador ha ganado o perdido.

**Lógica del Juego**:

* El programa debe tener una lista predefinida de palabras secretas y seleccionar una al azar para cada partida.
* Debe mostrar al jugador el "tablero", que consiste en guiones bajos representando las letras de la palabra secreta y una lista de las letras ya intentadas.
* El jugador tiene un número limitado de intentos (ej. 6 vidas).
* En cada turno, el jugador ingresa una letra. El programa debe validar si la entrada es una sola letra y si no ha sido intentada antes.
* Si la letra está en la palabra secreta, se revelan todas sus apariciones (ej. \_ \_ \_ \_ \_ -> \_ a \_ a \_). Si no está, el jugador pierde una vida.
* El juego termina cuando el jugador adivina todas las letras (gana) o se queda sin vidas (pierde).

**Conceptos integrados:** Lógica de juegos, random.choice, listas y/o sets (para letras adivinadas), manipulación de strings, bucles while, condicionales if/else, funciones para modularizar (ej. mostrar\_tablero(), validar\_entrada()), manejo de estado del juego.

**Ejercicio 15: Proyecto Final - Batalla Naval Simplificada**

Crea una versión del juego "Batalla Naval" en una cuadrícula de 5x5:

* El programa debe "esconder" un barco de 3 casillas en una fila o columna aleatoria.
* El jugador tiene 10 turnos para adivinar las coordenadas (ej. "A3") y hundir el barco.
* El programa debe gestionar el tablero (una lista de listas), validar la entrada del usuario, indicar si un disparo fue "Agua" o "Tocado", y llevar la cuenta de los turnos.
* Al final, debe declarar si el jugador ganó o perdió.

**Conceptos integrados:** Lógica de juegos, listas anidadas, funciones, bucles while y for, condicionales, random, manipulación de strings.