|  |  |
| --- | --- |
|  | Trabajo grupal o individual |

## Identificación del trabajo

|  |  |
| --- | --- |
| 1. **Módulo:** | 4 |
| 1. **Asignatura:** | Scripting |
| 1. **RA:** | Aplica estructuras de datos complejos, utilizando listas, tuplas y diccionarios, en la manipulación de datos |
| 1. **Docente Online:** | Gonzalo Cárdenas |
| 1. **Fecha de entrega**: | 01-08-2023 |
|  |  |
|  |  |

## Identificación del/los estudiante/s

|  |  |
| --- | --- |
| **Nombre y apellido** | **Carrera** |
| William Huera | Técnico en Data Science |
|  |  |
|  |  |
|  |  |
|  |  |

## Introducción

Se desempeña un papel clave en una empresa dedicada a la creación de software educativo escolar, enfocándote específicamente en el área de matemáticas. Actualmente, te han solicitado desarrollar la base de la nueva versión de este software mediante la implementación de scripts que incluyan una serie de operaciones básicas y mensajes de entrada y salida relacionados con el tema.

A continuación, te presento un ejemplo de script que servirá de referencia para abordar las siguientes preguntas

**Problemática**

## Desarrollo

**Item1**:

Analiza el uso de expresiones condicionales e iteradores para la llamada y el resultado esperado, a partir de un script, preparando un informe que dé cuenta del análisis realizado para las estructuras. Para ello

1. Identifica el o los elementos que pueden agregar inteligencia al programa (expresiones condicionales o iteradores) usados en la función signo\_numero, indicando la manera en que se usa en un mínimo de 3 líneas y un máximo de 5.
2. Identifica el o los elementos que pueden agregar inteligencia al programa (expresiones condicionales o iteradores) usados en la función es\_par, indicando la manera en que se usa en un mínimo de 3 líneas y un máximo de 5.
3. Identifica el o los elementos que pueden agregar inteligencia al programa (expresiones condicionales o iteradores) usados en la función n\_promedio, indicando la manera en que se usa en un mínimo de 3 líneas y un  
   máximo de 5.
4. Identifica el o los elementos que pueden agregar inteligencia al programa (expresiones condicionales o iteradores) usados en la función num\_multiplos, indicando la manera en que se usa en un mínimo de 3 líneas y un  
   máximo de 5.

|  |
| --- |
| ## codigo  **def** signo\_numero(): # se define la funcion signo\_numero  num **=** int (input()) #se solicita el ingreso de un numero por consola  **if** num **<** 0:  print()  **elif** num **==** 0:  print ()  **else**:  print ()    # 1: la primera condicion if num < 0, verifica si el numero ingresado es menor a 0,  # y si es verdadero, el programa imprime el número  # 2: la segunda condicion if num == 0, verifica si el numero ingresado es igual a 0,  # y si es verdadero, el programa imprime el número    **def** es\_par (): # se define la función es\_par  numero **=** int (input()) #se solicita el ingreso de un numero por consola, y se asigna a la variable numero  **if** numero **%** 2 **!=** 0:  print (**f**"{numero}es impar")  **else**:  print(**f**"{numero}es par")    # en la función se recibe un número desde consola, y lo almacena en la variable numero  # 1: la primera condición numero % 2 != 0, usa el operador modulo para obtener el residuo de  # la división del número por 2, si el residuo es diferente de 0 entonces el número es impar e imprime # el número  # 2: la segunda condición else, si el residuo es 0, entonces el número es par e imprime el número    **def** n\_promedio(): # se define la función n\_promedio  print("():")  nota **=** int (input ())  **while** nota **!=** **-**1:  total **=** total **+** nota  contar **=** contar **+** 1  print ("():")  nota **=** int (input ())  promedio **=** total **/** contar  print (""**+** str(promedio))    # en la función se recibe un integer desde consola, y lo almacena en la variable nota, y se  # calcula el promedio de las notas ingresadas  # 1: tenemos un iterador while nota != -1: que va recorrer cada nota y la va ir sumando en  # la variable total  # 2: también tenemos la variable contar donde se almacenan la cantidad de notas  # 3: en la variable promedio asignamos el cálculo de la suma total de notas divididas para la cantidad # de notas, y finalmente se imprime el promedio      **def** num\_multiplos(): # se define la funcion num\_multiplos  contador **=** 0 # se crea al vaiable contador y la iniciamos en 0  num **=** int (input(""))  **for** i **in** range (num):  **if** (i **%** num **==** 0):  contador **+=** 1  print("El numero "**+** str (num) **+** "posee" **+** str(contador) **+** "multiplos.\n")    # en la función se recibe un número desde consola, y lo almacena en la variable num  # tenemos un iterador for i in range (num): y las iteraciones están en función de la  # variable num  # preguntamos si el residuo es 0 (es múltiplo de num), si es múltiplo aumentamos el contador en 1  # y finalmente se imprime el valor del contador |
| GitHub**:** [**https://github.com/whuera/scripting/blob/master/modulo3/m4-t1.py**](https://github.com/whuera/scripting/blob/master/modulo3/m4-t1.py) |

**Item2**:

Determina una función matemática, que, dentro de su programación incluya el uso de estructuras condicionales e iteradores. Para ello, busca en internet un script de dicha función y realiza lo siguiente:

1. Copia el script de una función matemática, que no se encuentre incluida dentro de los ejercicios de la problemática, e indica la dirección desde donde se obtuvo.
2. Identifica la estructura condicional utilizada en el script.
3. Identifica la estructura iterativa utilizada en el script.
4. Describe cómo estas estructuras entregan inteligencia al código, entre 5 y 10 líneas.

|  |
| --- |
| **def** factorial(n): # definición de la función  **if** n **<** 0: # estructura condicional  **return** "El factorial no está definido para números negativos."  **elif** n **==** 0:  **return** 1  **else**:  result **=** 1  **while** n **>** 0: # uso de iterador  result **\*=** n  n **-=** 1  **return** result  # la función recibe un número entero, se evalua si el número es menor a 0, termina el programa  # y se imprime "El factorial no está definido para números negativos."  # si el número es igual a 0, se muestra Salida: 1 (0! = 1)  # si las dos condiciones anteriores no se cumplen, entonces se calcula el factorial del número ingresado  # impresion  print(factorial(5)) # Salida: 120 (5! = 5 \* 4 \* 3 \* 2 \* 1 = 120)  print(factorial(0)) # Salida: 1 (0! = 1)  print(factorial(**-**2)) # Salida: "El factorial no está definido para números negativos." |
| GitHub: <https://github.com/whuera/scripting/blob/master/modulo3/m4-t1-factorial.py> |

**Item3**:

Implementa una propuesta de software matemático mediante la creación de un script, identificando el uso de expresiones condicionales e iteradores. Este script debe recibir primero un valor *n* que indique la cantidad de notas a ingresar, validar que sea distinto de 0, ingresar las notas con un iterador, y finalmente calcular promedio de notas y desviación estándar validando si algún valor es cero.

Se evaluará lo siguiente:

1. Crear un script que use expresiones condicionales e iteradores, el script debe recibir un valor *n* que indica la cantidad de notas a ingresar, validando la condición n ≠ 0.
2. Ingresar con un iterador las *n* notas solicitadas.
3. Calcular el promedio de nota.
4. Calcular la desviación estándar.

## Conclusión:

Utilizar funciones y diccionarios en Python ofrece numerosas ventajas que facilitan el desarrollo y mantenimiento de programas.

**Ventajas de utilizar funciones:**

1. Reutilización de código: Las funciones te permiten encapsular bloques de código en tareas específicas y reutilizarlos en diferentes partes del programa. Esto fomenta la modularidad y evita la duplicación innecesaria de código.

2. Mantenibilidad: Al dividir el código en funciones más pequeñas y con propósitos específicos, el código general se vuelve más fácil de entender y mantener. Las actualizaciones y correcciones pueden aplicarse de manera más localizada.

3. Abstracción: Las funciones ocultan los detalles de implementación al usuario, lo que permite trabajar a un nivel más alto de abstracción. Esto hace que el código sea más fácil de usar y comprender.

4. Legibilidad: Al utilizar funciones con nombres descriptivos, el código se vuelve más legible y autodocumentado. Esto es especialmente útil cuando otros desarrolladores revisan o colaboran en el proyecto.

5. Depuración y pruebas: Al dividir el código en funciones, es más fácil aislar problemas y realizar pruebas unitarias, lo que simplifica la tarea de encontrar y corregir errores.

**Ventajas de utilizar diccionarios:**

1. Estructura clave-valor: Los diccionarios son ideales para almacenar y acceder a datos en función de claves únicas. Esta estructura es muy útil para organizar información relacionada.

2. Búsqueda eficiente: Los diccionarios en Python utilizan tablas hash, lo que permite búsquedas rápidas y eficientes por clave. Esto hace que sea muy útil para trabajar con grandes conjuntos de datos.

3. Flexibilidad: Los diccionarios pueden contener cualquier tipo de datos, incluso otros diccionarios, listas o funciones, lo que brinda una gran flexibilidad en la representación de datos complejos.

4. No requiere índices numéricos: A diferencia de las listas que se indexan mediante números, los diccionarios utilizan claves, lo que los hace más intuitivos y menos propensos a errores de índice.

5. Combinación con iteradores: Los diccionarios en Python son compatibles con iteradores, lo que permite recorrer fácilmente sus elementos para realizar operaciones en ellos.

En conclusión, el uso adecuado de funciones y diccionarios en Python contribuye a un código más estructurado, modular, reutilizable y fácil de mantener

## Bibliografía:

(w3schools, 2020)

Documentación guía, material de apoyo del módulo 3