# Trabajo grupal o individual

## Identificación del trabajo

a. **Módulo:** 3
b. **Asignatura:** Scripting
c. **RA:** Aplica estructuras de datos complejos, utilizando listas, tuplas y diccionarios, en la manipulación de datos.
d. **Docente Online:** Gonzalo Esteban Cárdenas Rubio
e. **Fecha de entrega**:

## Identificación del estudiante

| **Nombre y apellido** | **Carrera** |
|---|---|
| Endert Alejandro Guerrero Camacho | Data Science |

## Introducción

En el presente trabajo, se abordan diferentes aspectos relacionados con la programación en Python, mostrando una serie de funciones y scripts que ejemplifican su utilidad en diversas situaciones. A lo largo del informe, se exploran cuatro funciones: "suma()", "saluda()", "invierte()" y "completa()", cada una con un propósito específico y ejemplos prácticos. Además, se presentan dos scripts que demuestran la manipulación de listas, tuplas y diccionarios, así como el cálculo de estadísticas básicas. Python, un lenguaje versátil y de fácil comprensión, ofrece una amplia gama de herramientas que facilitan la resolución de problemas y la manipulación de datos.

También se presenta un escenario hipotético donde se deben aprovechar todos los componentes de Python y sus funcionalidades, en dicho escenario debemos crear un programa capaz de recibir la entrada de información para procesarla y así poder brindar el resultado deseado.

## Desarrollo

# ítem 1

*Función "suma()"*

```
def suma(a,b,c):
    resultado=a+b+c
    print("la suma es: ",resultado)


tp = 55, 55.22, 33.33
suma(*tp)
```

La función "suma" toma tres parámetros, 'a', 'b' y 'c', y calcula la suma de estos haciendo uso de una tupla llamada 'tp' para almacenar los valores.

Para pasar los valores almacenados en la tupla 'tp' como datos de la función "suma", se utiliza el operador de desempaquetado (*), este permite que los elementos de la tupla se descompongan en tres argumentos individuales, el resultado de la suma se almacena en la variable 'resultado' la cual se muestra en pantalla más adelante, es importante mencionar que las tuplas son estructuras de datos inmutables en Python, lo que significa que una vez creadas, no se pueden modificar sus elementos.

*Función "saluda()"*

```
def saluda(nombre, edad, sexo, nacionalidad):
    print("Hola",nombre)
    print("Tienes:", edad)
    print("eres del sexo", sexo)
```

```
    print("Eres de nacionalidad", nacionalidad)


dicc = {"nombre":"Juan", "edad":20, "sexo":"Masculino", "nacionalidad":"CL"}
saluda(**dicc)
```

La función "saluda" toma cuatro parámetros: 'nombre', 'edad', 'sexo' y 'nacionalidad', y muestra en pantalla un mensaje de saludo con los datos proporcionados.

El diccionario "dicc" contiene información sobre la persona, al pasar los valores de "dicc" como argumentos de la función "saluda", se utiliza el operador de desempaquetado de diccionario (**). Esto permite que los valores se distribuyan de forma correcta, el desempaquetado de diccionarios es una forma eficiente de pasar múltiples valores a una función utilizando sus claves como nombres de parámetros. Esto facilita la reutilización de la función con diferentes conjuntos de datos, ya que solo es necesario cambiar los valores del diccionario sin modificar la llamada a la función en sí.

*Función "invierte()"*

```
def invierte(lst):
    print(lst)
    lst_rev = lst[::-1]
    print(lst_rev)


lst = ["IPP", "Python", "Scripting", "Curso", "Material"]
invierte(lst)
```

La función "invierte" toma una lista como argumento, y su objetivo es imprimir la lista original y luego imprimir la misma lista, pero en orden inverso. en el ejemplo proporcionado, Cuando se llama a la función "invierte" con la lista "lst", la función imprime la lista original:

['IPP', 'Python', 'Scripting', 'Curso', 'Material']

Luego, la función crea una nueva lista llamada "lst_rev" que contiene los mismos elementos de la lista original, pero en orden inverso. Para lograr esto, utiliza la sintaxis de rebanado (slicing) con un paso de -1, que invierte el orden de los elementos en la lista:

['Material', 'Curso', 'Scripting', 'Python', 'IPP']

La función "invierte" imprime una lista dada y luego imprime la misma lista en orden inverso, lo que puede ser útil en diferentes situaciones cuando se requiere manipular el orden de los elementos en una lista.

*Función "completa"*

```
def completa(dct1,dct2):
    dct2.update(dct1)
    print(dct2)


dict1 = {"bookA" : 1, "bookB" : 2, "bookC" : 3}
dict2 = {"bookC" : 2, "bookD" : 4, "bookE" : 5}
completa(dict1,dict2)
```

En la función "completa" se utilizan dos diccionarios como estructuras de datos pares (clave-valor), "dct1" y "dct2", también se utiliza el método "update()" de los diccionarios para combinar los elementos de "dct1" con el diccionario "dct2". El método "update()" actualiza el diccionario "dct2", sobrescribiendo o agregando las claves y valores de "dct1". Es decir, el diccionario "dct2" se actualiza para incluir los elementos de "dct1". Si coinciden claves en ambos diccionarios los valores del diccionario "dct2" se reemplazan por los valores de "dct1". En conclusión, la función "completa" utiliza diccionarios como estructura de datos para combinar y actualizar sus elementos de acuerdo con los argumentos proporcionados.

# Ítem 2

*Script 1*

```
mylist = list((1,2,3,4))
print(mylist)
print(type(mylist))


mytuple = tuple(mylist)
print(mytuple)
print(type(mytuple))
```

En este script, se presentan una lista y una tupla. Las listas son modificables después de su creación, mientras que las tuplas una vez creadas, no pueden modificarse.

1. Variables de entrada:

- La entrada inicial es una lista la cual más adelante es convertida a tupla en la variable "mytuple".

2. Transformaciones o conversiones:

- La lista "mylist" se convierte en una tupla utilizando la función tuple() y se almacena en la variable mytuple.

3. Variables de salida:

- El script imprime tanto "mylist" como su tipo utilizando print(). Luego, imprime "mytuple" y su tipo también con print().

El script tiene como objetivo ilustrar cómo convertir una lista en una tupla, esta misma conversión se podría hacer de forma contraria usando list().

*Script 2*

```
lista1 = list(("nombre", "edad", "dirección"))

lista2 = list(("Juan", "30", "Valparaíso"))

clave_tupla = tuple(lista1)

valor_tupla = tuple(lista2)


dicc = dict(zip(clave_tupla, valor_tupla))

print(dicc)

print(type(dicc))
```

1. Variables de entrada:

- lista1: Es una lista que contiene las claves para crear un diccionario. En este caso, contiene los elementos "nombre", "edad" y "dirección".
- lista2: Es una lista que contiene los valores pares de las claves para crear un diccionario. En este caso, contiene los elementos "Juan", "30" y "Valparaíso".

2. Transformaciones o conversiones:

- Se convierten las listas "lista1" y "lista2" en tuplas utilizando la función incorporada tuple(). Esto garantiza que las claves y los valores sean inmutables y, por lo tanto, sean adecuados para utilizarlos como elementos de un diccionario. Las nuevas variables creadas son:
- clave_tupla: Una tupla que contiene las claves de la lista1, es decir, ("nombre", "edad", "dirección").

- valor_tupla: Una tupla que contiene los valores pares, es decir, ("Juan", "30", "Valparaíso").

3. Variables de salida:

- dicc: Es un diccionario que se crea utilizando las tuplas "clave_tupla" y "valor_tupla" con la función zip(). La función zip() combina ambas tuplas en pares clave-valor, que luego se utilizan para crear el diccionario. Luego se imprime el contenido del diccionario "dicc". También se imprime el tipo de la variable "dicc" utilizando la función type(). Esto muestra el tipo de dato de "dicc", que será 'dict'

El script realiza las transformaciones necesarias para crear un diccionario a partir de las listas "lista1" y "lista2", y luego imprime el diccionario resultante junto con su tipo. La estructura del diccionario final contiene las claves "nombre", "edad" y "dirección", con los valores correspondientes "Juan", "30" y "Valparaíso", respectivamente.

# Ítem 3

Importamos las librerías correspondientes para poder ejecutar el código correctamente.

```
import math
import pandas as pd
```

Calculamos los promedios de las notas, sumamos las notas y las dividimos según la cantidad de notas que tengan los estudiantes.

```
def cal_prom(notas):
    return sum(notas) / len(notas)
```

Calculamos la varianza para cada valor x en la lista "notas", se resta el promedio "prom", y el resultado se eleva al cuadrado con (x - prom) ** 2., se agrega "for x in notas" para decirle al programa que los valores de "X" se encuentran en "notas", ya que tenemos 3 notas por estudiante debemos sumar los resultados de la operación anterior entre si, para finalmente dividirlos según la cantidad de notas de cada estudiante.

Finalmente usaremos "math.sqrt(var)" que nos dará la desviación estándar como resultado.

```
def cal_var(notas):
```

```
    prom = cal_prom(notas)
    var = sum([(x - prom) ** 2 for x in notas]) / len(notas)
    return math.sqrt(var)
```

A partir de ahora crearemos el área principal del código, también creare un diccionario vacío donde de alojaran los estudiantes y sus notas.

```
def main() :
    dicc_notas = {}
```

Solicitamos al usuario ingresar, primeramente, el número de estudiantes y el número de notas que debería tener cada estudiante.

```
num_estudiantes = int(input("Ingrese el número de estudiantes: "))
num_notas = int(input("Ingrese el número de notas que presentan los
estudiantes: "))
```

Haciendo uso de "range" delimitamos la cantidad de veces que se repetirá el bucle, esta cantidad estará dictada por el número de estudiantes que ingresemos, mientras que el bucle donde ingresaremos las notas se repetirá también cuantas veces hallamos delimitado al momento de colocar cuantas notas presentará cada estudiante.

```
for i in range(num_estudiantes):
    estudiante = input(f"Ingrese el nombre del estudiante {i+1}: ")
    notas = []
    for n in range(num_notas):
        nota = float(input(f"Ingrese la nota {n+1} de {estudiante}: "))
        notas.append(nota)
    dicc_notas[estudiante] = notas
```

Ahora crearemos un diccionario que estará compuesto por las listas resultantes de las operaciones siguientes.

```
resultados = {
```

```
    "Estudiante": [],
    "Promedio": [],
    "Varianza": []
}
```

Por último calcularemos las estadísticas para cada estudiante y se agregaran los resultados al diccionario anterior.

```
for estudiante, notas in dicc_notas.items():
    promedio = cal_prom(notas)
    varianza = cal_var(notas)
    resultados["Estudiante"].append(estudiante)
    resultados["Promedio"].append(promedio)
    resultados["Varianza"].append(varianza)
```

Creamos un dataframe para poder observar de mejor manera los resultados

```
df = pd.DataFrame(resultados)
print(df)
```

Endert_Guerrero_T1_M3_Scripting.ipynb | item3.py

item3.py > ...

```python
#importamos las librerias necesarias
import math
import pandas as pd

# Función para calcular el promedio de notas
def cal_prom(notas):
    return sum(notas) / len(notas)

# Función para calcular la varianza de notas
def cal_var(notas):
    prom = cal_prom(notas)
    var = sum([(x - prom) ** 2 for x in notas]) / len(notas)
    return math.sqrt(var)

# Crear un diccionario vacío para almacenar los estudiantes y sus notas
dicc_notas = {}

# Solicitar al usuario ingresar el numero de estudiantes y el numero de notas que deberia tener cada estudiante.

num_estudiantes = int(input("Ingrese el número de estudiantes: "))
num_notas = int(input("Ingrese el número de notas que presentan los estudiantes: "))

for i in range(num_estudiantes):
    estudiante = input(f"Ingrese el nombre del estudiante {i+1}: ")
    notas = []
    for n in range(num_notas):
        nota = float(input(f"Ingrese la nota {n+1} de {estudiante}: "))
        notas.append(nota)
    dicc_notas[estudiante] = notas

# Crear un diccionario que estara compuesto por las listas resultantes de las siguentes operaciones.

resultados = {
    "Estudiante": [],
    "Promedio": [],
    "Varianza": []
}

# Calcular las estadisticas para cada estudiante y agregar los resultados al diccionario
for estudiante, notas in dicc_notas.items():
    promedio = cal_prom(notas)
    varianza = cal_var(notas)
    resultados["Estudiante"].append(estudiante)
    resultados["Promedio"].append(promedio)
    resultados["Varianza"].append(varianza)

# mostrar los resultados a traves de un DataFrame
df = pd.DataFrame(resultados)
print(df)
```

```
Ingrese el número de estudiantes: 5
Ingrese el número de notas que presentan los estudiantes: 4
Ingrese el nombre del estudiante 1: Endert
Ingrese la nota 1 de Endert: 5
Ingrese la nota 2 de Endert: 4
Ingrese la nota 3 de Endert: 3
Ingrese la nota 4 de Endert: 7
Ingrese el nombre del estudiante 2: Alejandro
Ingrese la nota 1 de Alejandro: 7
Ingrese la nota 2 de Alejandro: 6
Ingrese la nota 3 de Alejandro: 2
Ingrese la nota 4 de Alejandro: 1
Ingrese el nombre del estudiante 3: Manuel
Ingrese la nota 1 de Manuel: 7
Ingrese la nota 2 de Manuel: 7
Ingrese la nota 3 de Manuel: 1
Ingrese la nota 4 de Manuel: 1
Ingrese el nombre del estudiante 4: Jose
Ingrese la nota 1 de Jose: 1
Ingrese la nota 2 de Jose: 1
Ingrese la nota 3 de Jose: 3
Ingrese la nota 4 de Jose: 4
Ingrese el nombre del estudiante 5: Lalo
Ingrese la nota 1 de Lalo: 7
Ingrese la nota 2 de Lalo: 6
Ingrese la nota 3 de Lalo: 4
Ingrese la nota 4 de Lalo: 5
  Estudiante  Promedio  Varianza
0     Endert      4.75  1.479020
1  Alejandro      4.00  2.549510
2     Manuel      4.00  3.000000
3       Jose      2.25  1.299038
4       Lalo      5.50  1.118034
```

## Extra

Adjunto material extra en el IPYNB en el que trabaje mientras hacia la actividad.

## Conclusión

Las funciones y scripts en Python permiten comprender la flexibilidad y potencia que este lenguaje ofrece para resolver diferentes tareas. Las funciones, como "suma()", "saluda()", "invierte()" y "completa()", demuestran cómo se pueden crear bloques de código reutilizables y eficientes para realizar tareas específicas. Por otro lado, los scripts 1 y 2 muestran cómo trabajar con estructuras de datos como listas, tuplas y diccionarios, permitiendo manipular la información de manera sencilla. Gracias a todo esto podemos realizar programas eficientes y rápido como el que creamos anteriormente, para este aprovechamos todos los componentes aprendidos mientras se realizaban los ítems 1 y 2 demostrando así la capacidad de manejo de datos que ofrece el lenguaje.

## Bibliografía


- NumPy Developers. (2022, 18 diciembre). *Numpy.random.uniform — NumPY v1.25 Manual*. NumPy. Recuperado 1 de agosto de 2023, de https://numpy.org/doc/stable/reference/random/generated/numpy.random.uniform.html#numpy.random.uniform
- The pandas development team. (2020, febrero). *Pandas.DataFrame — Pandas 2.0.3 Documentation*. Pandas. Recuperado 31 de julio de 2023, de https://pandas.pydata.org/docs/reference/api/pandas.DataFrame.html?highlight=dataframe#pandas.DataFrame
- The pandas development team. (2020, febrero). *pandas.DataFrame.groupby — Pandas 2.0.3 Documentation*. Pandas. Recuperado 31 de julio de 2023, de

https://pandas.pydata.org/docs/reference/api/pandas.DataFrame.groupby.html#pandas.DataFrame.groupby

- W3.CSS. (s. f.). *Python Dictionary Update() Method*. Recuperado 31 de julio de 2023, de https://www.w3schools.com/python/ref_dictionary_update.asp
- W3.CSS. (s. f.). *Python zip() Function*. Recuperado 31 de julio de 2023, de https://www.w3schools.com/python/ref_func_zip.asp
- W3.CSS. (s. f.). *Python - Slicing Strings*. Recuperado 31 de julio de 2023, de https://www.w3schools.com/python/python_strings_slicing.asp
- W3.CSS. (s. f.). *Python round() Function*. Recuperado 31 de julio de 2023, de https://www.w3schools.com/python/ref_func_round.asp#
- Diaz, D. (2023, 18 julio). *¿Cómo usar los operadores de desempaquetado (*, **) en Python?* Geekflare. Recuperado 1 de agosto de 2023, de https://geekflare.com/es/python-unpacking-operators/