	Carátula para entrega de prácticas
Facultad de Ingeniería	Laboratorio de docencia

Laboratorios de computación salas A y B

Profesor: TISTA GARCÍA EDGAR

Asignatura: ESTRUCTURA DE DATOS Y ALGORITMOS I

Grupo: 1

No de Práctica(s): Guía práctica de estudio 10: Introducción a Python (II).

Integrante(s): MURRIETA VILLEGAS ALFONSO

Semestre: 2018 - 2

Fecha de entrega: 30 DE ABRIL DE 2018

Observaciones:

CALIFICACIÓN: _____

INTRODUCCIÓN A PYTHON (II)

INTRODUCCIÓN

Python es un lenguaje de alto nivel, caracterizado por ser multi-paradigma, además, de ser un lenguaje de programación que se caracteriza por ser interpretado y cuya filosofía hace hincapié en una sintaxis donde se favorece al código legible.

La presente práctica tiene entre sus muchos objetivos empezar a utilizar estructuras de control tanto selectivas como las condiciones if, else, etc y estructuras de control repetitivas como son los ciclos for o while. Además, pretende darnos a conocer las ventajas que ofrece Python a su comunidad como es la gran variedad de bibliotecas y cómo éstas pueden usarse con la finalidad de ampliar las posibilidades y creatividad al realizar programas por parte de los desarrolladores.

OBJETIVOS DE LA PRÁCTICA

- Aplicar estructuras de control selectivas
- Aplicar estructuras de control repetitivas
- Usar las bibliotecas estándar
- Generar una gráfica
- Ejecutar un programa desde la ventana de comandos
- Pedir datos al usuario al momento de ejecutar un programa

DESARROLLO

1. Antecedentes

En la práctica anterior se nos acercó a las características básicas del lenguaje de programación Python, esto con el fin de conocer poco a poco un lenguaje que sobretodo está pensado en dar una gran facilidad y legibilidad al momento de escribir y hacer códigos.
Como se vio, la gran mayoría de diferencias que existen con respecto al lenguaje C se dan en la parte sintáctica del lenguaje ya que como bien se sabe, Python está pensado en que sea amigable con los programadores al momento de escribir código, por otro lado , una de las mayores diferencias con respecto a C es sobretodo que en este lenguaje no hay necesidad de declarar variables ni determinar el tipo de dato que serán es por ello que se utilizan las correspondientes funciones para realizar las operaciones o acciones sobre estos dato.

Actividad 1. Análisis del manual de prácticas

Para poder hacer un mejor análisis de toda la parte teórica del manual de prácticas, se decidió separar en distintos sub-apartados.

1.1 Estructuras de control selectivas

Dentro de los lenguajes de programación una de las principales características que tienen es la posibilidad de condicionar acciones dentro de los algoritmos programados, esto es posible gracias al uso principalmente de 2 condiciones, "if" y "else" , sin embargo, cada lenguaje se diferencia por un manejo distinto de estas condiciones además de la forma sintáctica en la que se citan.

1. If

La condición "If" por sí solo sirve para ejecutar partes de código y dependiendo del resultado de la condición que se encuentra dentro de su argumento se realizan o no. A continuación se muestra un programa donde podemos ver como es la parte sintáctica además del uso de esta condición (Ver imagen 1).

```
"""
MURRIETA_VILLEGAS_ALFONSO
1_EJERCICIO / PRÁCTICA_10
"""
def DatesCompare(par1,par2,par3):

    if par1<par2:
      print('{} es mayor que {}'. format(par2,par1))

    if par1 == par2 == par3:
      print('TRUE')
    1

def main():

    par1= eval(input("Ingrese valor 1: "))
    par2= eval(input("Ingrese valor 2: "))
    par3= eval(input("Ingrese valor 3: "))
    DatesCompare(par1,par2,par3)#Poner los argumentos

main()
```

Imagen 1: Uso de "if" para comparaciones

NOTA: Algo importante dentro de las condiciones en Python es que podemos poner no solamente 2 argumentos para comparar (Ver imagen 1, ovalo amarillo).

2. If – Else

Como se mencionó anteriormente la condición "if" sirve para comparar datos, sin embargo, en dado caso de que no se cumpla la condición del if existen 2 posibilidades, la primera es que simplemente termine esa parte del programa o que en dado caso entre en una opción alternativa de código que en este caso sería el bloque de código dentro de la condición "else" (Ver imagen 2).

```
def DatesCompare(par1,par2,par3):

    if par1<par2:
        print('El mayor es: {} '. format(par2))
    else:
        print('El mayor es: {} '. format(par1))
```

Imagen 2: Uso de "if - else" para comparaciones.

3. If- elif- Else

Este tipo de condición sirve para generar más de una comparación y sobre todo si es del mismo tipo, con respecto al lenguaje C, elif sería los casos dentro de una condición "Switch".

En este tipo de condición la manera en que se enuncian las condiciones es la siguiente:

- El primer caso solamente se agrega un "if"
- En los "n" casos se utiliza "elif"
- Y el último caso, se utiliza "else"

A continuación, se muestra un ejemplo de uso de condiciones "elif" dentro de un programa de comparaciones de 3 números, donde se trata de encontrar el número más grande.

```
"""
MURRIETA_VILLEGAS_ALFONSO
1_EJERCICIO / PRÁCTICA_10
"""
def DatesCompare(par1,par2,par3):

    if par1>par2:
        if par1>par3:
            print('El primero es el mayor')

    elif par2>par3:
        print('El segundo es el mayor')

    else:
        print('El tercero es el mayor')

def main():

    par1= eval(input("Ingrese valor 1: "))
    par2= eval(input("Ingrese valor 2: "))
    par3= eval(input("Ingrese valor 3: "))
    DatesCompare(par1,par2,par3)#Poner los argumentos

main()
```

Imagen 3: Uso de "if - elif - else" para comparaciones.

Como se puede ver en el código anterior también se puede anidar condiciones, en el caso anterior se hace para poder determinar si el primer dato ingresado es el mayor (Encerrado en amarillo).

NOTA: Algo totalmente diferente con respecto a C es la forma sintáctica en la que podemos evitar el escribir la condición "elif", ya que dentro de una condición if podemos buscar un dato de la forma " if *Variable_de_comparación* in (*1,2,3,4*): "

COMPARACIÓN ENTRE PYTHON Y C

Realmente podemos ver que dentro de Python se pueden realizar las mismas comparaciones que en el lenguaje C, sin embargo, las diferencias de estas se encuentran en la parte sintáctica, por ejemplo, lo que se conocía como "Switch" se lleva a través de "elif", otro ejemplo es el uso de comparaciones mediante los ciclos "if _ in (_,_,_)".

1.2 Estructuras de control repetitivas

1. Ciclo while

Un ciclo es la manera de ejecutar una o varias acciones repetidamente. A diferencia de las estructuras IF o IF-ELSE que sólo se ejecutan una vez. Para que el ciclo se ejecute, la condición siempre tiene que ser verdadera.

```
6  def main():
7
8      n = int(input("Escriba un número pequeño: "))
9      u = int(input("Escriba un número grande: "))
10     print("Los números intermedios son:")
11     while n < u:
12         n=n+1
13         print(n)
14
15 main()
```

Imagen 4: Ciclo while dentro de un programa de Python.

2. Ciclo for

Los ciclos for a diferencia de los ciclos while se caracterizan por tener bien definidos el inicio y fin de las iteraciones que se realizarán dentro de estos. En el caso de Python se utilizan comúnmente para realizar iteraciones dentro de listas y diccionarios.

Iteración en listas

Las listas como se vio en prácticas anteriores son un tipo de Dato Abstracto que nos ofrece la ventaja de poder redimensionar y reubicar el dato empleado, dicho de otra forma, son mutables.

```
6  def main():
7      n = 0
8
9      for n in [1,2,3,4,5]:
10         print(n)
11
12     for n in range(5): #Lo que hacemos es determinar el rago o sea hasta 5
13         print(n)
14
15     #Podemos determinar el inicio y fin del rango o intervalo
16     for n in range(-2,2):
17         print(n)
18
19         #NOTA: El último elemento no entra en el rango
20     for num in ["Uno", "Dos", "Tres", "cuatro" ]:
21         print(num)
22
23 main()
```

```
In [30]: runfile('C:/Users/pon_c/Downloads/Documents/Programación/
Python/EDA-1/2_practice/04_manual.py', wdir='C:/Users/pon_c/Downloads/
Documents/Programación/Python/EDA-1/2_practice')
1
2
3
4
5
0
1
2
3
4
-2
-1
0
1
Uno
Dos
Tres
cuatro
```

Imagen 5: Código y salida de un programa en Python empleando for para la iteración de elementos de una lista.

Iteración en diccionarios

Los Diccionarios también llamados **matrices asociativas**, ya que son colecciones que contiene *clave*:*valor* por ejemplo:

diccionario = {"enero":1,"febrero":2,"marzo":3}

En la parte de la clave podemos usar cualquier tipo de valor inmutable: números, cadenas, booleanos o tuplas menos listas o diccionarios dado que son mutables. A continuación, se muestra una imagen de cómo se utilizan los diccionarios dentro de Python (La imagen fue recopilada del manual de prácticas)

```
#Creando un diccionario
elementos = { 'hidrogeno': 1, 'helio': 2, 'carbon': 6 }

for llave, valor in elementos.items():
    print(llave, " = ", valor)
```

```
helio  =  2
carbon  =  6
hidrogeno  =  1
```

```
#Obteniendo sólo las llaves
for llave in elementos.keys():
    print(llave)
```

```
helio
carbon
hidrogeno
```

```
#Obteniendo sólo los valores
for valor in elementos.values():
    print(valor)
```

```
2
6
1
```

Imagen 6: Código y salidas empleando un diccionario en Python.

NOTA: La diferencia que existe entre diccionarios con las listas o las tuplas es que los valores almacenados en el diccionario no se le accede a través de su índice si no a través de su clave utilizando el operador de [] así que de esta forma nuestros diccionarios no tienen un orden prestablecido

También Python a diferencia de muchos lenguajes de programación no necesita crear un índice para ir enumerando las iteraciones realizadas, para esto podemos utilizar la función enumarate() de la siguiente forma (Ver imagen 7).

```
#Si se necesita iterar utilizando un índice
for idx, x in enumerate(elementos):
    print("El índice es: {} y el elemento: {}".format(idx, x))
```

```
El índice es: 0 y el elemento: helio
El índice es: 1 y el elemento: carbon
El índice es: 2 y el elemento: hidrogeno
```

Imagen 7: Código y salidas empleando un diccionario en Python y utilizando la función enumarate()..

Por último, algo que podemos realizar dentro de estos ciclos es la disponibilidad de la condición "else" una vez que se ha terminado el ciclo, cabe destacar que no debe existir un break o algo que interrumpa el ciclo.

COMPARACIÓN ENTRE PYTHON Y C

Realmente en esta parte creo que es donde más diferencias existen con respecto con C, para empezar la manera en que se llevan las iteraciones es muy distintas pues primero no se debe declarar un índice, ni iterador, segundo, existen funciones encargadas de llevar a cabo distintas acciones dentro de las iteraciones como es el caso de enumarate(), además el tipo de datos que se pueden utilizar dentro de estos ciclos y condiciones es muy distintos como es el caso de los diccionarios.

1.3 Bibliotecas

Una de las ventajas que nos ofrece Python es la amplia variedad de bibliotecas que la comunidad ha creado con el propósito de ampliar las posibilidades y facilitar la construcción de algoritmos dentro de Python. Todas las funcionalidades de Python son proporcionadas a través de bibliotecas que se encuentran en la colección de The Python Standard Library, la mayoría de estas bibliotecas son multi-plataforma. A continuación, se muestra el uso de la biblioteca "math" para realizar operaciones trigonométricas.

```
6 import math
7
8 def main():
9
10     x= math.cos(math.pi)
11     print(x)
12
13 main()
```

```
In [32]: runfile('C:/Users/pon_c/Downloads/Documents/Programación/
Python/EDA-1/2_practice/05_manual.py', wdir='C:/Users/pon_c/Downloads/
Documents/Programación/Python/EDA-1/2_practice')
-1.0
```

Imagen 8: Uso de la función math

Algunas de las funciones más comunes dentro de la comunidad científica de Python son las siguientes (Ver tabla):

Biblioteca	Funciones y características
NumPy (Numerical Python)	Es una de las bibliotecas más populares de Python, es usado para realizar operaciones con vectores o matrices de una manera eficiente. Contiene funciones de Álgebra Lineal, transformadas de Fourier, generación de números aleatorios e integración con Fortran, C y C++.
SciPy (Scientific Python)	Es una biblioteca hace uso de Numpy y es utilizada para hacer operaciones más avanzadas como transformadas discretas de Fourier, Álgebra Lineal, Optimización, etc.
Matplotlib	Esta biblioteca es usada para generar una veriedad de gráficas en 2D y 3D, donde cada una de las configuraciones de la gráfica es programable. Se puede usar comando de Latex para agregar ecuaciones matemáticas a las gráficas.
Scikit Learn (Machine Learning)	Ésta biblioteca está basada en los anteriores y contiene algoritmos de aprendizaje de máquina, reconocimiento de patrones y estadísticas para realizar clasificación, regresión, clustering, etc.
Pandas (Manipulación de datos)	Esta biblioteca es utilizada para manipulación de datos, contiene estructuras de datos llamadas data frames que se asemejan a las hojas de cálculo y a los cuales se le puede aplicar una gran cantidad de funciones

Tabla 1: Principales bibliotecas de Python.

NOTA: Una de las opciones que nos ofrece Python es la ventaja de solamente importar las funciones que necesitamos de las bibliotecas a través del uso de **"from *Biblioteca* import *funcion1*, *funcion2*"**.

Además, para omitir el prefijo de la biblioteca de donde se está llamando la función simplemente debemos hacer lo siguiente **"from *Biblioteca* import *funcion1*, *funcion2"***.

1.4 Graficación

Matplotlib como se mencionó en el apartado anterior, es una biblioteca usada para generar gráficas 2D y 3D, donde cada una de las configuraciones o parámetros de la gráfica es programable.
A continuación, se muestra un ejemplo del uso de esta función.

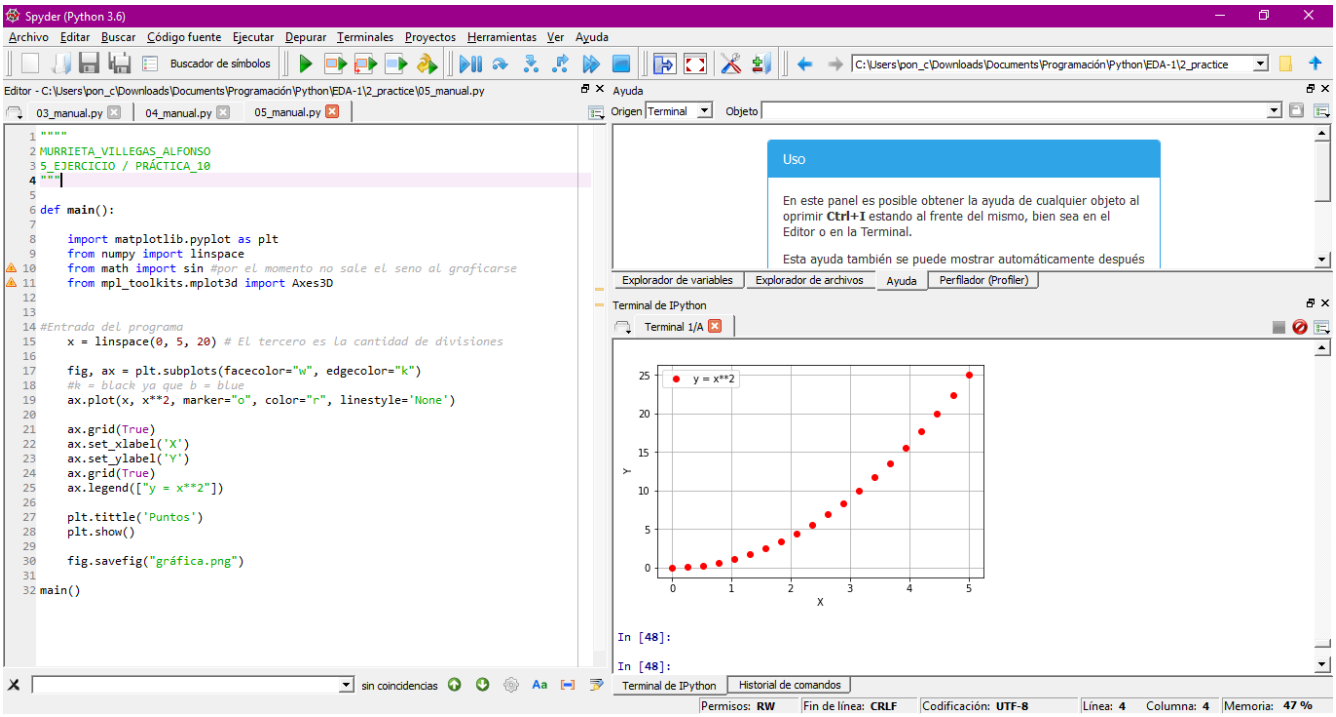

Imagen 9: Captura de pantalla de un programa donde se muestra del lado izquierdo el código fuente y del lado derecho la salida del programa que es una gráfica de una función cuadrática.

1.5 Ejecución desde ventana de comandos

Todo el código que se ha visto hasta el momento puede ser guardado en archivos de texto plano con la extensión '.py'. Para ejecutarlo desde la ventana de comandos se escribe el comando:

python nombre_archivo.py

Al igual que en otros lenguajes, también se puede se le puede pedir al usuario que introduzca ciertos datos de entrada cuando se ejecute un programa, sin embargo, esto no es posible dentro de "Jupyter"

NOTA: Debido a que dentro de Python no declaramos el tipo de dato que estamos usando para ello se utiliza de manera indirecta un casteo de la siguiente forma:

Num= int(input('Opción 1:'))

Donde la parte anterior a la función input se encarga de convertir la variable, en la práctica pasada lo que se realizó fue el uso de una función "eval()" que se encarga de asociar a la variable un tipo de dato.

2. Programa de comparación de contraseña

Para esta actividad lo que se nos pide es realizar un programa de comparación de contraseñas, debido a que la contraseña puede tener caracteres y números el tipo de dato que se debe guardar es una cadena ya que esta nos da la posibilidad de guardar la contraseña con esas características. Por otro lado, y como ayuda para nosotros, el entorno de programación o IDE que se permitió usar es Spyder.

1. Primera parte

En este apartado solamente se pide que no permita continuar al usuario a menos de que escriba correctamente la contraseña, para ello lo que se utilizó fue un ciclo "while" donde se ejecute un bloque de código pidiendo nuevamente la contraseña solamente en caso de que se incorrecta la contraseña, esta comparación se hace en el argumento del while "***while contra != contrareal:***". A continuación se muestra la salida del programa.

```
In [1]: runfile('C:/Users/pon_c/Downloads/Documents/
Programación/Python/EDA-1/2_practice/01_practica.py',
wdir='C:/Users/pon_c/Downloads/Documents/Programación/
Python/EDA-1/2_practice')

Ingrese la contraseña: hola
What do you want bro ? (Wrong password)

Ingrese la contraseña: adios
What do you want bro ? (Wrong password)

Ingrese la contraseña: jajaSalu2
La contraseña es correctar
```

Imagen 10: Salida del programa en el primer apartado

2. Segunda parte

Para este apartado se nos pedía que el usuario solo dispusiera de una cantidad fija de oportunidades al ingresar la contraseña, la solución más fácil y sencilla fue meter todo el código anterior dentro de un ciclo for ya que en este podemos fijar la cantidad de oportunidades del usuario, por otro lado, dentro del ciclo for debe existir 2 condiciones la primera que es un if nos permite reingresar la contraseña solamente si se ha ingresado incorrectamente dentro de la cantidad de oportunidades ya establecidas, la segunda condición es un else el cual en caso de que sea correcta la contraseña nos mande el mensaje de que es correcto.

```
In [2]: runfile('C:/Users/pon_c/Downloads/Documents/
Programación/Python/EDA-1/2_practice/02_practica.py',
wdir='C:/Users/pon_c/Downloads/Documents/Programación/
Python/EDA-1/2_practice')
Sólo 5 intentos

Ingrese la contraseña: hola
What do you want bro ? (Wrong password)

Ingrese la contraseña: adios
What do you want bro ? (Wrong password)

Ingrese la contraseña: hola
What do you want bro ? (Wrong password)

Ingrese la contraseña: jajaSalu2
La contraseña es correctar
```

Imagen 11: Salida del programa en el segundo apartado, se ingresan 3 veces incorrectamente la contraseña y posteriormente se ingresa correctamente.

NOTA: Es importante poner un break dentro de ese else debido a que si no se escribe esto seguiría iterando dentro del ciclo for

3. Tercera parte

En este apartado se nos pedía que mediante una función aparte se pudiera regresar un valor booleano dependiendo de si el usuario había ingresado incorrectamente la contraseña o si la había ingresado correctamente (A través del mismo valor booleano que podía ser False o True).

Lo primero que se hizo fue pasar todo el código anterior dentro de una función aparte, posteriormente se creó una función principal donde solamente debíamos realizar la parte de impresión de los resultados y lo que le enviaba la función externa (La de comparación y el valor de la variable booleana).

Dentro de la función de comparación lo que se hizo fue crear otra variable de tipo booleano la cual se encargaría de determinar si era falso o cierto lo ingresado por el usuario.

Para iterar y limitar las oportunidades del usuario se utilizó un ciclo for condicionado a una cantidad de intentos, dentro de este ciclo for se encontraban 2 bloques de código principales. El primer bloque mediante una condición if si la contraseña era incorrecta se daba la posibilidad de volver a ingresar la contraseña, mientras que en el else solamente se regresaba a la función principal el valor booleano "true".

A continuación, se muestra la salida de nuestro programa.

```
In [11]: runfile('C:/Users/pon_c/Downloads/Documents/
Programación/Python/EDA-1/2_practice/03_practica.py',
wdir='C:/Users/pon_c/Downloads/Documents/Programación/
Python/EDA-1/2_practice')
Sólo 5 intentos

Ingrese la contraseña: hola
What do you want bro ? (Wrong password)
False

Ingrese la contraseña: adios
What do you want bro ? (Wrong password)
False

Ingrese la contraseña: jajaSalu2
Contraseña correcta
True
```

Imagen 12: Salida del programa en el tercer apartado, se muestra 2 veces la contraseña incorrectamente y en el último caso se muestra la salida con respecto a la contraseña correcta.

NOTA: En la primera versión de este ejercicio hubo un pequeño detalle dentro del *else*, ya que al utilizar un break en ese momento termina toda la función, por lo tanto, en esta segunda versión, se agregó un *return* previamente.

3. Calificaciones alumno

Para este ejercicio se pidió que un alumno ingresara las calificaciones de sus tareas y exámenes, las condiciones principales son que solamente debe agregar 5 tareas y 4 exámenes, las condiciones implícitas son como es obvio que las calificaciones estén entre un rango de 0 a 10.

Lo primero que se hizo fue utilizar una función encargada de dar los promedios tanto de los exámenes como de las tareas, para ello lo que los datos ingresados se guardaban dentro de una lista.

NOTA: Esta función para determinar el tamaño de la extensión se necesitaba pasar el valor de la extensión de la lista (Esto ya estaba previamente asignado en variables correspondientes a las listas).

```
12      while cont != limite:
13          dato=eval(input('Ingrese la calificación: ') )#Eval para ver cada
14          if dato >= 0 and dato <=10:
15              lista.append(dato)
16              cont=cont+1
17          else:
18              print('Ingrese un dato válido')
19
20      for i in range(limite):
21          prom = prom+lista[i]
22      prom=prom/limite
23      print('Promedio: {}'.format(prom))
24      return prom
```

Imagen 13: Parte de código encargado del guardado y de la comparación de la entrada de las calificaciones.

Por último, para poder determinar cuándo determinar el .5 extra de calificación o cuando quitar .5 al promedio lo que se utilizó fue otra función a la cual se les pasó tanto el promedio del examen y el promedio de tareas.

```
def resultado(calTareas, calExam):

    if calTareas > 8.5:
        calExam=calExam+0.5
        print('Se sumó .5, su promedio es: {}'.format(calExam))

    elif  7.0 < calTareas <= 8.5 :
        calExam=calExam
        print('No hubo cambios, su promedio es: {}'.format(calExam))

    else:
        calExam=calExam-0.5
        print('Se restó .5, su promedio es: {}'.format(calExam))
```

Imagen 13: Parte de código encargado de la asignación del punto extra

A continuación, se muestra la salida del programa ya finalizado:

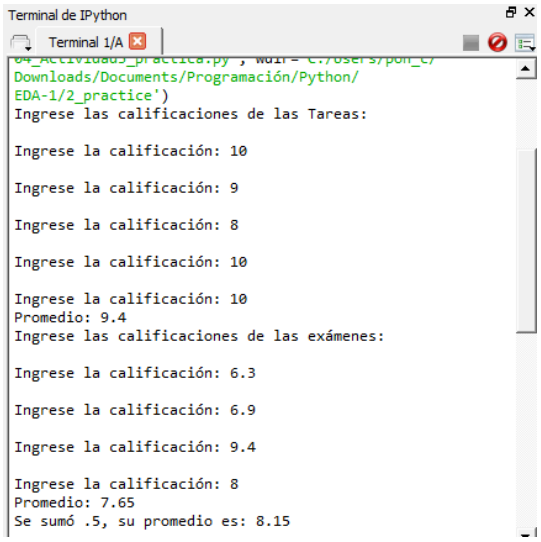

Imagen 14: Salida del programa de la actividad 3, se muestra la asignación de las calificaciones y como al final debido al promedio se suma .5

RELACIÓN ENTRE EJERCICIOS Y TEMA

Como se pudo ver a lo largo de esta práctica, todos los ejercicios realizados tuvieron relación con Python en concreto en el uso de condiciones, sin embargo, también hubo problemas que estaban pensados en darnos algunas herramientas extras sobre todo para aprovechar las ventajas que ofrece este lenguaje de programación sobre otros.

Por ejemplo, una gran herramienta es la disponibilidad de tener un lenguaje orientado a objetos y como de esta forma y a través de bibliotecas podemos realizar aplicaciones matemáticas, como fue el caso de la graficación de una función a través de **Matplotlib**.

Por otra parte, Python nos ofrece la facilidad de poder escribir y emplear funciones para poder hacer programas mucho más modulares y óptimos como fueron todos los ejercicios de la práctica donde se puede apreciar que el uso de funciones hace mucho más fácil el entender y escribir nuestros programas.

Por último, el último ejercicio fu sin duda la culminación de todo lo aprendido en Python hasta e momento, ya que en este debemos no solamente utilizar las bases aprendidas en la anterior práctica, sino que a través del conocimiento previo de C y adaptándolo a Python pude realizar un programa realmente modular y muy fácil de entender.

CONCLUSIONES

Durante esta práctica se conocieron más a profundidad las características y la forma sintáctica de las condiciones y ciclos que se emplean en Python, además, se dieron algunos ejemplo y ventajas que ofrece este lenguaje con respecto a otros como es C. Como mencioné en la práctica anterior, Python tiene la ventaja que al ser moderno sabe que carencias tenían lenguajes pasados, además de que su visión está

dirigida a un público que está destinado a hacerlo crecer, es por ello que el simplemente ver la variedad de bibliotecas y funciones que nos ofrece es sin duda asombroso.

En el primer envío pude concluir toda la actividad 2 excepto el último apartado esto debido a que al momento de escribir los bloques de condición se me olvidó escribir un *return* dentro de un *else* (Checar apartado donde describo con más detalle el error). Además, lamentablemente no pude llevar a cabo la tercera actividad, sin embargo, como es notable el código ya terminado realmente es algo que me enorgullece entrega sobre todo porque es un código que realmente es modular y para ser sincero yo no me hubiera visto programando algo tan bien hecho cuando entre a EDA-1, tal vez en este momento no sea el mejor código, pero creo que poco a poco voy mejorando mis capacidades sobretodo el cómo abstraer los problemas.

Por último, para resolver la última actividad me basé un poco en lo clase teórica de construcción de algoritmos y es que para poder llevar a cabo la construcción del algoritmo tuve que visualizar desde un punto general todo lo que se nos pedía y como de esta forma podía generar módulos encargados de cada condición o requisito del programa.

REFERENCIAS

The Python Language Reference. Recuperado el 20 de abril de 2018, de https://docs.python.org/3/reference/index.html

Cuevas Álvarez. (2016). Python 3 Curso Práctico. Ra – Ma.

Mark J. Guzdial, Barbara Ericson. (2015). *Introducción a la computación y programación con Python*. 3ra Edición. Madrid España.

Solano Jorge A. (2017). *Manual de Prácticas del laboratorio de Estructuras de Datos y algoritmos 1*. UNAM, Facultad de Ingeniería.