	Carátula para entrega de prácticas
Facultad de Ingeniería	Laboratorio de docencia

Laboratorios de computación salas A y B

Profesor:	M.I. Marco Antonio Martínez Quintana.
Asignatura:	Estructura de Datos y Algoritmos I.
Grupo:	17
No de Práctica(s):	10
Integrante(s):	Acosta Rodríguez Eder Alberto.
No. de Equipo de cómputo empleado:	
No. de Lista o Brigada:	
Semestre:	2020-2
Fecha de entrega:	14 De Abril de 2020
Observaciones:	

CALIFICACIÓN: _____

Guía Práctica de Estudio 09: Introducción a Python (II).

OBJETIVO:

Aplicar las bases del lenguaje de programación Python en el ambiente de Jupyter notebook.

ACTIVIDADES:

- 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.

INTRODUCCIÓN:

Python es un lenguaje de programación **versátil multiplataforma y multiparadigma** que se destaca por su código legible y limpio. Una de las razones de su éxito es que cuenta con una licencia de código abierto que permite su utilización en cualquier escenario. Esto hace que sea uno de los **lenguajes de iniciación** de muchos programadores siendo impartido en escuelas y universidades de todo el mundo. Sumado a esto cuenta con grandes compañías que hacen de este un uso intensivo.

if

La declaración IF sirve para ejecutar código dependiendo del resultado de una condición.

```
def obtenerMayor(param1,param2):
    if param1 < param2:
        print('{} es mayor que {}'.format(param2, param1))
```

```
obtenerMayor(5, 7)
```

```
7 es mayor que 5
```

```
obtenerMayor(7, 5)   #No imprime nada
```

Se puede encadenar más de una una condición sin tener que agregar un operador booleano.

```
x = y = z = 3
if x == y == z:
    print(True)
```

```
True
```

if-else

Este tipo de declaraciones se usan para dar una opción en el caso de que la condición no se cumpla.

```
def obtenerMayorv2(param1,param2):
    if param1 < param2:
        return param2
    else:
        return param1
```

```
print ("El mayor es {}".format( obtenerMayorv2(4, 20) ))
```

```
El mayor es 20
```

```
print ("El mayor es {}".format( obtenerMayorv2(11, 6) ))
```

```
El mayor es 11
```

Para comparaciones simples, Python no tiene un operador ternario (x ? True : False), pero se puede emular con if-else:

```
def obtenerMayor_idiom(param1,param2):
    #La variable valor va a tener el valor de param2 is el if es verdadero
    #de lo contrario tendra el valor de param1
    valor = param2 if (param1 < param2) else param1
    return valor
```

```
print ("El mayor es {}".format( obtenerMayor_idiom(11, 6) ))
```

```
El mayor es 11
```

if-elif-else

Este tipo de declaraciones sirve para generar varios casos de prueba. En otros lenguajes es similar a case o switch.

```
def numeros(num):
    if num==1:
     print ("tu numero es 1")
    elif num==2:
     print ("el numero es 2")
    elif num==3:
     print ("el numero es 3")
    elif num==4:
     print ("el numero es 4")
    else:
     print ("no hay opcion")
```

```
numeros(2)
```

```
el numero es 2
```

```
numeros(5)
```

```
no hay opcion
```

En alunos casos, se puede evitar la repetición de código del if-elif-else de la siguiente manera:

```
def numeros_idiom(num):
    #La tupla tiene las opciones válidas
    if num in (1,2,3,4):
        print("tu numero es {}".format(num))
    else:
        print ("{} no es una opcion".format(num))
```

```
numeros_idiom(2)
```

```
tu numero es 2
```

```
numeros_idiom(5)
```

```
5 no es una opcion
```

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.

```
#Ejemplo 1
def cuenta(limite):
    i = limite
    while True:
        print (i)
        i = i -1
        if i == 0:
            break # Rompiendo el ciclo
```

```
cuenta(10)
```

```
10
9
8
7
6
5
4
3
2
1
```

```
#Ejemplo 2
def factorial(n):
    i = 2
    tmp = 1
    while i <n+1:
        tmp = tmp * i
        i = i + 1
    return tmp
```

```
print (factorial(4))
```

```
24
```

```
print (factorial(6))
```

```
720
```

Ciclo for

Este ciclo es el más común usado en Python, se utiliza generalmente para hacer iteraciones en una lista, diccionarios y arreglos.

Iteración en listas

```
for x in [1,2,3,4,5]:
    print(x)
```

```
1
2
3
4
5
```

```
#La función range() sirve para generar una lista
for x in range(5): #este caso es equivalente a range(0,5)
    print(x)
```

```
0
1
2
3
4
```

```
#También se puede inizializar desde números negativos
for x in range(-5,2):
    print(x)
```

```
-5
-4
-3
-2
-1
0
1
```

```
for num in ["uno", "dos", "tres", "cuatro"]:
    print(num)
```

```
uno
dos
tres
cuatro
```

Iteración en diccionarios

```
#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
```

En algunos lenguajes de programación se crea un índice para iterar un conjunto de elementos (for (int i=0; i < elementos.size(); ++i)), sin embargo con Python se puede utilizar la función enumerate() en su lugar.

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

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

Los ciclos for pueden hacer uso del else una vez que terminan de iterar, pero no funciona si se rompe el ciclo.

```
def cuenta_idiom(limite):
    for i in range(limite, 0, -1):
        print(i)
    else: #Corresponde al for, NO al IF
        print("Cuenta finalizada")
```

```
cuenta_idiom(5)
```

```
5
4
3
2
1
Cuenta finalizada
```

```python
#Se rompe el ciclo y la sentencia else del for no se ejecuta
def cuenta_idiomv2(limite):
    for i in range(limite, 0, -1):
        print(i)
        if i == 3:
            break  #Se rompe el ciclo
    else: #Corresponde al FOR, NO al IF
        print("Cuenta finalizada")
```

```python
cuenta_idiomv2(5)
```

```
5
4
3
```

Bibliotecas

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.

Referencia del lenguaje: https://docs.python.org/3/reference/index.html
Bibliotecas estándar: https://docs.python.org/3/library/

```python
#Para utilizar una biblioteca, ésta se debe de importar
import math

x = math.cos(math.pi)

print(x)
```

```
-1.0
```

```python
#También se pueden importar todas las funciones de la bibliotecas, de esta manera no se tiene que usar el prefijo
#de la biblioteca, que en el ejemplo anterior fue math
from math import *

x = cos(pi)  #No se utiliza el prefijo math

print(x)
```

```
-1.0
```

```python
#Otra manera es importar sólo las funciones que se necesitan
from math import cos, pi

x = cos(pi)

print(x)
```

```
-1.0
```

Bibliotecas más usadas

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++.
Fuente: http://www.numpy.org/

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.
Fuente: http://www.scipy.org/

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.
Fuente: http://matplotlib.org/

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.
Fuente: http://scikit-learn.org/

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. Fuente: http://pandas.pydata.org/

ANEXO 1: En esta guía se explica de manera más detallada el uso de las bibliotecas Numpy y Matplotlib.

Jupyter Notebook GitHub:
https://github.com/eegkno/FI_UNAM/blob/master/02_Estructuras_de_datos_y_algoritmos_1/Anexos/Anexo_I.ipynb

Jupyter Notebook Visualizador:
http://nbviewer.jupyter.org/github/eegkno/FI_UNAM/blob/master/02_Estructuras_de_datos_y_algoritmos_1/Anexos/Anexo_I.ipynb

Graficación

Matplotlib (http://matplotlib.org/) es una biblioteca usada para generar gráficas en 2D y 3D, donde cada una de las configuraciones de la gráfica es programable. En el siguiente ejemplo se mostrará la configuración básica de una gráfica.

EL API de matplotlib se encuentra en http://matplotlib.org/api/index.html

```
#Esta línea se ocupa para que las gráficas que se generen queden embebidas dentro de la página
%pylab inline
```

```
#Importando las bibliotecas
import matplotlib.pyplot as plt
from mpl_toolkits.mplot3d import Axes3D
```

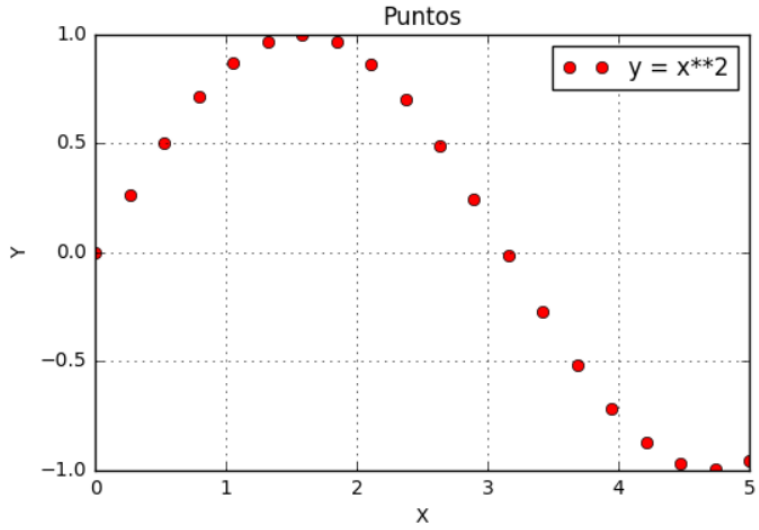
```
#Datos de entrada
x = linspace(0, 5, 20) #Generando 10 puntos entre 0 y 5
```

```
fig, ax = plt.subplots(facecolor='w', edgecolor='k')
ax.plot(x, sin(x), marker="o",color="r", linestyle='None')

ax.grid(True)
ax.set_xlabel('X')  #Etiqueta del eje x
ax.set_ylabel('Y')  #Etiqueta del eje y
ax.grid(True)
ax.legend(["y = x**2"])

plt.title('Puntos')
plt.show()

fig.savefig("gráfica.png") #Guardando la gráfica
```



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
```

Entrada de datos

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. Esto no se puede hacer desde la notebook, ya que los datos se introducen en las celdas que se van agregando a lo largo de la página, tal y como se ha venido manejando hast ahora. Como ejemplo se va a ejecutar el archivo lectura_datos.py desde una ventana de comandos.

```
python lectura_datos.py
```

Al momento de ejecutar el programa, se va a pedir al usuario que introduzca su nombre, esto se logra con el siguiente código:

```
#Se pide el nombre al usuario
print ("Hola, ¿cómo te llamas?")
#Se leen los datos introducidos por el usuario y se asignan a la
variable nombre
nombre = input()
#Se escribe el nombre solicitado
print ("Buen día {}".format(nombre)
```

Después de esto se despliega un menú donde se indican las operaciones que puede realizar el usuario, una vez que indicada la operación, se solicitan los datos necesarios para ejecutarla.

```
print ("---Calculadora---") #Opciones para el usuario
print ("1- Sumar")
print ("2- Restar")
print ("3- Multiplicar")
print ("4- Dividir")
print ("5- Salir")
```

En la siguiente línea se solicita que el usuario especifique alguna de las operaciones, a diferencia de la primera petición, la función input() ahora tiene una cadena que se le despliega al usuario. A su vez, los datos que recibe la función input() son de tipo string, por lo que se tienen que transformar a entero con la función int() para poder realizar operaciones aritméticas.

```
op = int(input('Opcion: '))
```

DESARROLLO Y ACTIVIDADES:

Código de la práctica:

Obtener_Mayor.py - C:\Users\Eder Berno Acosta\Documents\U.N.A.M\Facultad de Ingeniería\Estructuras de Datos y Algoritmos I\Práctica 10 EDA_Python\Obtener_Mayor.py (3.8.2)

File Edit Format Run Options Window Help

```
def obtenerMayor(param1,param2):
    if param1<param2:
        print('{} es mayor que {} '.format(param2,param1))
```

Obtener_Mayor_v2.py - C:\Users\Eder Berno Acosta\Documents\U.N.A.M\Facultad de Ingeniería\Estructuras de Datos y Algoritmos I\Práctica 10 EDA_Python\Obtener_Mayor_v2.py (3.8.2)

File Edit Format Run Options Window Help

```
def obtenerMayorv2(param1,param2):
    if param1<param2:
        return param2
    else:
        return param1
```

Obtener_Mayor_idiom.py - C:\Users\Eder Berno Acosta\Documents\U.N.A.M\Facultad de Ingeniería\Estructuras de Datos y Algoritmos I\Práctica 10 EDA_Python\Obtener_Mayor_idiom.py (3.8.2)

File Edit Format Run Options Window Help

```
def obtenerMayor_idiom(param1,param2):
    #La variable valor va a tener el valor de param2 is el if es verdadero
    #de lo contrario tendra el valor de param1
    valor = param2 if(param1<param2)else param1
    return valor
```

Numeros.py - C:/Users/Eder Berno Acosta/Documents/U.N.A.M/Facultad de Ingeniería/Estructuras de Datos y Algoritmos I/Práctica 10 EDA_Python/Numeros.py (3.8.2)

File Edit Format Run Options Window Help

```
def numeros(num):
    if num==1:
        print("tu numero es 1")
    elif num==2:
        print("el numero es 2")
    elif num==3:
        print("el numero es 3")
    elif num==4:
        print("el numero es 4")
    else:
        print("no hay opcion")
```

Numeros_idiom.py - C:/Users/Eder Berno Acosta/Documents/U.N.A.M/Facultad de Ingeniería/Estructuras de Datos y Algoritmos I/Práctica 10 EDA_Python/Numeros_idiom.py (3.8.2)

File Edit Format Run Options Window Help

```
def numeros_idiom(num):
    #La tupla tiene las opciones válidas
    if num in(1,2,3,4):
        print("tu numero es {}".format(num))
    else:
        print("{} no es una opcion".format(num))
```

Cuenta.py - C:/Users/Eder Berno Acosta/Documents/U.N.A.M/Facultad de Ingeniería/Estructuras de Datos y Algoritmos I/Práctica 10 EDA_Python/Cuenta.py (3.8.2)

File Edit Format Run Options Window Help

```
#Ejemplo 1
def cuenta(limite):
    i = limite
    while True:
        print(i)
        i = i -1
        if i ==0:
            break #Rompiendo el ciclo
```

Factorial.py - C:/Users/Eder Berno Acosta/Documents/U.N.A.M/Facultad de Ingeniería/Estructuras de Datos y Algoritmos I/Práctica 10 EDA_Python/Factorial.py (3.8.2)

File Edit Format Run Options Window Help

```
#Ejemplo 2
def factorial(n):
    i = 2
    tmp = i
    while i<n+1:
        tmp = tmp * i
        i = i +1
    return tmp
```

For_X.py - C:\Users\Eder Berno Acosta\Documents\U.N.A.M\Facultad de Ingeniería\Estructuras de Datos y Algoritmos I\Práctica 10 EDA_Python\For_X.py (3.8.2)

File Edit Format Run Options Window Help

```
for x in [1,2,3,4,5]:
    print(x)
```

Range_For_X.py - C:\Users\Eder Berno Acosta\Documents\U.N.A.M\Facultad de Ingeniería\Estructuras de Datos y Algoritmos I\Práctica 10 EDA_Python\Range_For_X.py (3.8.2)

File Edit Format Run Options Window Help

```
#fila función range () sirve para generar una lista
for x in range(5): #este caso es equivalente a renge(0,5)
    print(x)
```

Range_For_X_Negativo.py - C:\Users\Eder Berno Acosta\Documents\U.N.A.M\Facultad de Ingeniería\Estructuras de Datos y Algoritmos I\Práctica 10 EDA_Python\Range_For_X_Negativo.py (3.8.2)

File Edit Format Run Options Window Help

```
#También se puede inicializar desde números negativos
for x in range(-5,2):
    print(x)
```

For_Num.py - C:\Users\Eder Berno Acosta\Documents\U.N.A.M\Facultad de Ingeniería\Estructuras de Datos y Algoritmos I\Práctica 10 EDA_Python\For_Num.py (3.8.2)

File Edit Format Run Options Window Help

```
for num in ("uno","dos","tres","cuatro"):
    print(num)
```

For_Elementos.py - C:\Users\Eder Berno Acosta\Documents\U.N.A.M\Facultad de Ingeniería\Estructuras de Datos y Algoritmos I\Práctica 10 EDA_Python\For_Elementos.py (3.8.2)

File Edit Format Run Options Window Help

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

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

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

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

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

Cuenta_idiom.py - C:\Users\Eder Berno Acosta\Documents\U.N.A.M\Facultad de Ingeniería\Estructuras de Datos y Algoritmos I\Práctica 10 EDA_Python\Cuenta_idiom.py (3.8.2)

File Edit Format Run Options Window Help

```
def cuenta_idiom(limite):
    for i in range(limite,0,-1):
        print(i)
    else: #Corresponde al for, NO al IF
        print("Cuenta finalizada")
```

Math_bibliotecas.py - C:\Users\Eder Berno Acosta\Documents\U.N.A.M\Facultad de Ingeniería\Estructuras de Datos y Algoritmos I\Práctica 10 EDA_Python\Math_bibliotecas.py (3.8.2)

File Edit Format Run Options Window Help

```
#para utilizar una biblioteca, ésta se debe de importar
import math

x = math.cos(math.pi)

print(x)

#También se puede importar todas las funciones de las bibliotecas, de esta manera no se tiene que usar el prefijo
#de la biblioteca, que en el ejmplo anterior fue math
from math import *

x = cos(pi) #No se utiliza el prefijo math

print(x)

#Otra manera es importar sólo las funciones que se necesitan
from math import cos, pi

x = cos(pi)

print(x)
```

Grafica.py - C:\Users\Eder Berno Acosta\Documents\U.N.A.M\Facultad de Ingeniería\Estructuras de Datos y Algoritmos I\Práctica 10 EDA_Python\Grafica.py (3.8.2)

File Edit Format Run Options Window Help

```
#Esta línea se ocupa para que las gráficas que se generen queden embedidas dentro de la página
%pylab inline

#importando las bibliotecas
import matplotlib.pyplot as plt
from mpl_toolkits.mplot3d import Axes3D

#Datos de entrada
x = linscape(0,5,20) #Generando 10 puntos entre 0 y 5

fig, ax = plt.subplots(facecolor='w', edgecolor='k')
ax.plot(x, sin(x), maker="o", color="r", linestyle="None")

ax.grid(True)
ax.set_xlabel('X') #Etiqueta el eje x
ax.set_ylabel('Y') #Etiqueta el eje y
ax.grid(True)
ax.legend(["y = x**2"])

plt.tittle('Puntos')
plt.show()

fig.savefig("Gráfica.png") #Guardando la gráfica
```

CONCLUSIONES:

Gracias a esta práctica se comprendió la manera correcta de utilizar las estructuras condicionales en este lenguaje de programación por la manera en que es de manera más sencilla la declaración de ellas y así como su implementación.

BIBLIOFRAGÍA:

- Tutorial oficial de Python: https://docs.python.org/3/tutorial/
- Galería de notebooks: https://wakari.io/gallery
- Matplotlib: http://matplotlib.org/