Tutorial de Python con Jupyter Notebook

Ing. Diego Quisi

Objetivo: Aprender a realizar programas simples en Python utilizando cuadernos de Jupyter.

Conocimientos previos: Conocimientos de programación básica: variables, estructuras de control, funciones, matrices y clases.

Python

Python es un lenguaje de alto nivel, multiparadigma y con tipado dinámico.

Si bien se usa en varios ámbitos, recientemente se ha convertido en el lenguaje más utilizado para programación científica, junto con las librerías **NumPy** (matrices), **Matplotlib** (visualizar datos) y otras.

El tutorial no asume conocimiento de Python, pero tampoco explica el lenguaje en detalle.

Cuadernos de Jupyter Notebook

La forma tradicional de correr un programa en python es con el comando `python nombre.py`, donde `nombre.py` es un archivo con código fuente python.

En lugar de eso, para este curso utilizaremos un servidor de Jupyter Notebook con cuadernos de código. Estos *cuadernos* (*notebooks*) nos permiten combinar texto y código, organizados en *celdas*, lo cual es más cómodo para probar cosas nuevas y documentar lo que hacemos.

El servidor de cuadernos se inicia ejecutando `jupyter notebook` desde la línea de comandos.

Si tenemos cuadernos para abrir, antes de correr ese comando debemos ir al directorio con los cuadernos, de modo de poder abrirlos después. El servidor corre continuamente mientras usamos los cuadernos.

Una vez que el servidor corre y se abre el navegador, se elege abrir un cuaderno anterior o crear uno nuevo. Luego, se escribe y ejecuta texto y código en el cuaderno, y podés guardar el estado de un cuaderno con `ctrl+s` en cualquier momento. Se guarda tanto el código como el resultado de las ejecuciones.

La guía de instalación disponible para que puedas correr python y jupyter en tu computadora.

Uso de Cuadernos de Jupyter

Los cuadernos tienen dos tipos de celdas, de código y de texto. La celda que estás leyendo es una celda de texto escrita con Markdown, un lenguaje de marcado parecido al que utiliza wikipedia para sus páginas o al HTML.

Las celdas de código son *ejecutables*, es decir, se pueden correr individualmente (con `ctrl+enter` o desde el menu `Cell -> Run Cells`)

In [ ]:

```
#Este es un comentario porque empieza con #

#Esta es una celda de código.

#Se ejecuta con ctrl+enter. Probalo.

#La función print puede imprimir varias cosas
print("Hola Mundo desde Jupyter Notebook") #impresión de un string
print(5) # impresión de un número

#Intentá imprimir el string "IMAGENES":
```

Python básico

Las variables en python no necesitan ser declaradas, simplemente se definen al ser utilizadas por primera vez. Además, (si bien no es recomendable) pueden cambiar de tipo volviendo a definir.

In [ ]:

```
x="hola"
print(x)

x=5
print(x)

y=x+2.5
print(y)
```

Tipos de datos básicos

Python tiene los mismos datos básicos que otros lenguajes: enteros, flotantes, strings y booleanos. Además, las listas son un tipo predefinido en el lenguaje.

Numeros

Python tiene soporte para números enteros y de punto flotante.

In [ ]:

```
### Enteros ###

x = 3

print("- Tipo de x:")
print(type(x)) # Imprime el tipo (o `clase`) de x
print("- Valor de x:")
print(x)       # Imprimir un valor
print("- x+1:")
print(x + 1)   # Suma: imprime "4"
print("- x-1:")
print(x - 1)   # Resta; imprime "2"
print("- x*2:")
print(x * 2)   # Multiplicación; imprime "6"
print("- x^2:")
print(x ** 2)  # Exponenciación; imprime "9"
# Modificación de x
x += 1
print("- x modificado:")
print(x)  # Imprime "4"

x *= 2
print("- x modificado:")
print(x)  # Imprime "8"

print("- Varias cosas en una línea:")
print(1,2,x,5*2) # imprime varias cosas a la vez
```

In [ ]:

```
### Flotantes ###

y = 2.5
print("- Tipo de y:")
print(type(y)) # Imprime el tipo de y
print("- Varios valores en punto flotante:")
print(y, y + 1, y * 2.5, y ** 2) # Imprime varios números en punto flotante
```

Booleanos

Python implementa todos los operadores usuales de la lógica booleana, usando palabras en inglés (`and, or, not`) en lugar de símbolos (||, &&, !, etc)

También tiene los típicos operadores de comparación: `<,>,>=,<=,==,!=`

In [ ]:

```
### Booleanos ###

v1 = True #el valor verdadero se escribe True
v2 = False #el valor verdadero se escribe False

print("- Valores de v1 y v2:")
print(v1,v2)

print("- Tipo de v1:")
print(type(v1)) # Imprime la clase de un valor booleano ('bool')

print("- v1 and v2:")
print(v1 and v2) # y lógico; imprime False
print(v1 or v2)  # o lógico; imprime True
print(not v1)   # negación lógica, imprime False

print(3 == 5)  # Imprime False ya que son distintos
print(3 != 5)  # Imprime True ya que son distintos
print(3 < 5)  # Imprime True ya que 3 es menor que 5
```

Listas

Python tiene soporte para listas como un tipo predefinido del lenguaje. Para crear una lista basta con poner cosas entre `[]` (corchetes) y separarlas con `,` (comas).

In [ ]:

```
print("- Lista con 4 números:")
a=[57,45,7,13] # una lista con cuatro números
print(a)

print("- Lista con 3 strings:")
b=["hola","chau","buen día", 5] # una lista con tres strings
print(b)

# la función `len` me da la longitud de la lista
print("- Longitud de la lista:")
n=len(a)
print(n)
```

In [ ]:

```
#Para acceder a sus elementos, se utiliza el []
# Los índices comienzan en 0
print("- Elemento con índice 0 de la lista:")
print(b[0])
print("- Elemento con índice 1 de la lista:")
print(b[1])
print("- Elemento con índice 2 de la lista:")
print(b[2])
print("- Elemento ultimo de la lista:")
print(b[-1])
```

In [ ]:

```
# para crear una lista vacía, (sin elementos), simplemente ponemos []
vacia=[]
print("Lista vacía:")
print(vacia)

# También podés crear una sub-lista o slice especificando un rango de indices, el rango superior e
s no inclusivo
print("- Elementos del índice 0 al 1 (2-1):")
print(a[0:2])
print("- Elementos del índice 1 al 3 (4-1):")
print(a[1:4])
#Si ponés nada antes del : se asume que pusiste 0
print("- Elementos desde el comienzo al indice 1 (2-1) :")
print(a[:2])
#Si no ponés nada después del : se asume que tomás todos hasta el final
print("- Elementos desde el indice 1 hasta el final:")
print(a[1:])
```

```
#Si no pones nada ni antes ni después es como tomar todo
print("- Todos los elementos:")
print(a[:])
print(a)


#Si el fin es igual al comienzo, es un rango vacío, por ende se obtiene una lista vacía
print("- Rango vacío -> lista vacía:")
print(a[2:2])
#Rangos negativos
print(' - Rango negativo - > los 3 ultimos elementos')
print(a[-3:])
```

Una lista es un objeto

Python permite definir clases y crear objetos de esas clases, pero esos temas están fuera de este tutorial. No obstante, una lista es un objeto, y tiene varios métodos. Entre ellos está el método `append`, que permite agregar un elemento a la lista. Los métodos se invocan de la siguiente forma `objeto.metodo(parametro1,parametro2,...)`. Adicionalmente se tiene algunos metodos como: pop, extends, push, map, etc.

In [ ]:

```
#por último, le podés agregar elementos a una lista con el método `append`
print("- Una lista con 3 strings:")
a=['una','lista','de', 5, 5.6]
print(a)

print("- La misma lista luego de agregarle un string más:")
a.append('ultimo')
print(a)
lista=[3,5,4,8.9,10]
print(lista)
print(" - Sumar la lista de elementos con el metodo sum")
print(sum(lista))
```

Tuplas

Las tuplas son como las listas, pero no se pueden modificar. Son como unas listas de sólo lectura. Se crean con `()` (paréntesis) en lugar de `[]` (corchetes).

In [ ]:

```
#Podés crear una tupla con valores entre () separados por ,
a=(1,2,57,4)
print("- Una tupla de cuatro elementos:")
print(a)
print("- El elemento con índice 2:")
print(a[2])
print("- Los elementos entre los índices 0 y 2:")
print(a[0:2])

# la siguiente línea genera un error de ejecución
#a.append(28)
```

Diccionarios

Los diccionarios en python nos permite almacenar listas de tipo clave - valor, en donde los datos pueden ser de cualquier tipo de dato y la clave no se puede repetir.

In [ ]:

```
diccionario = {'nombre': 'Diego', 'apellido': 'Quisi', 'edad': 29, 'materias': ['Programacion', 'Si
stemas Expertos', 'IA 1']}
print(diccionario)
print(diccionario['nombre']) # imprime Diego
print(diccionario['materias'][0:2])
```

Estructuras de control

En Python no hay llaves (*{}*) ni *begin...end* para marcar el comienzo y fin de un bloque, sino que eso se logra con la indentación. La indentación por defecto son 4 espacios en blanco.

Entonces va a ser necesario indentar correctamente para utilizar sentencias `if`, `for` o para definir funciones.

if

El `if` es como el de otros lenguajes, pero no pide paréntesis y termina con `:`. Su sintaxis es:

```
if condicion :
    cuerpo del if (indentado con 4 espacios)
else:
    cuerpo del else (indentado con 4 espacios)
```

In [ ]:

```
edad = 25

print("La persona es")
if edad < 18: # el if termina con : para indicar donde acaba la condición
    # el print va indentado con 4 espacios para indicar que está dentro del cuerpo del if
    print("Menor")
else:
    #Lo mismo con este print
    print("Mayor")

print("De edad")
```

In [ ]:

```
#Ejercicio 1
# Pasar a escala de grises el color codificado en los elementos de la lista `pixel`

pixel= [0.6,0.3,0.4] # intensidades de cada canal.
#El elemento 0 es el R, el 1 el G y el 2 el B

# la intensidad en escala de grises es el promedio de la intensidad de cada canal R, G y B
intensidad=0 # IMPLEMENTAR

print("La intensidad es:")
print(intensidad)
```

In [ ]:

```
#Ejercicio 2
# Pasar a blanco y negro el valor de intensidad codificado en la variable intensidad


# podemos considerar que un pixel se convierte en blanco si su intensidad en escala de grises es m
ayor a 0.5
# y negro de lo contrario
bw = 0 # IMPLEMENTAR

print("En blanco y negro el pixel sería: (0 -> negro, 1 -> blanco)")
print(bw)
```

Estructuras de repeticion

For

Los `for` son parecidos a los `if`, pero tienen la sintaxis `for variable in lista:`. En este caso, `variable` es la variable que va a ir cambiando, y `lista` es una lista de python (o un `iterable` que es parecido)

que va a ir cambiando, y `lista` es una lista de python (o un `iterable` que se parezca)

In [ ]:

```
print("- Impresion de los elementos de la lista:")

# Imprimir los strings de mi_lista por separado
mi_lista=["img","python","numpy"]
for s in mi_lista:
    print(s)# este print va con indentación

#calcular la suma de los números e imprimirla
suma=0
mis_numeros=[5,8,17,12]
for numero in mis_numeros:
    suma+=numero
print("- La suma de los números es:")
print(suma)
```

Cuando no tenemos una lista y queremos hacer un for "común" y que la variable que cambia sea un número que va incrementándose, podemos utilizar la función `range`.

In [ ]:

```
#un for de 0 a 3, para imprimir esos valores
print("Un for de 0 a 3")
for i in range(5):
    print(i)

#En Python los índices comienzan en 0, y por eso los rangos también.


#También se puede comenzar el rango en otro valor en lugar de 0
print("- Un for de 2 a 5:")
for j in range(2,6):
    print(j)

print(" - Un for negativo: ")
for k in range(-1,-5,-1):
    print(k)
```

In [ ]:

```
#Ejercicio 3: Escribir un for para buscar el máximo de la lista e imprimirlo
lista=[44,11,15,29,53,12,30]
maximo=0
# IMPLEMENTAR
for numero in lista:
    if numero>maximo:
        maximo=numero
# debe imprimir 53
print("- El maximo es:")
print(maximo)
```

In [ ]:

```
#Ejercicio 4: Escribir un for para buscar el minimo elemento de la lista e imprimir su _posición_
lista=[44,11,15,29,53,12,30]
posicion=0
# IMPLEMENTAR
for numero in lista:
    if numero>maximo: #<minimo >maximo
        maximo=numero
posicion=lista.index(maximo)
#debe imprimir 4
print("- La posición del máximo es:")
print(posicion) #para seguir la indicacion imprimira 4

# Ejercicio 5 : Ordenar la lista de forma asendente
#IMPLEMENTAR
orden=sorted(lista)
print("- Lista ordenada:")
print(orden)
```

While

Los bucles `while` permite ejecutar ciclos, o bien secuencias periódicas que nos permiten ejecutar código múltiples veces.

In [ ]:

```
suma, numero = 0, 1
while numero <= 10:
    suma += numero
    numero += 1

print ("La suma es " + str(suma))
```

Funciones

Las funciones se definen con la palabra clave `def` y tienen la sintaxis `def nombre_funcion(parametros):`. Para devolver un valor utilizamos la palabra clave `return`. Una funcion puede o no retornar un valor

In [ ]:

```
#esta funcion recibe dos números y devuelve su suma

def sumar(a,b):
    return a+b


c=sumar(2,5)
print("2+5=")
print(c)


#esta funcion recibe una lista y devuelve la suma de sus elementos
def sumar_todos(lista):
    suma=0
    for v in lista:
        suma+=v
    return suma

mi_lista=[54,12,99,15]
print("los elementos de la lista suman:")
print(sumar_todos(mi_lista))

# Ejercicio 7

# Crear una funcion en donde me permita enviar como parametro el numero de elementos y
# devolver un listado de la serie fibonassi con el numero de elementos ingresado.


#IMPLEMENTAR

def fib(n):
    lista=[]
    a, b = 0, 1
    while n > 0:
        a, b = b, a + b
        n -= 1
        lista.append(a)
    return lista
# Driver Program
print("Lita de numeros fibonassi")
print(fib(8))
```

In [ ]:

```
#Ejercicio 6
# Escribir una función que reciba una lista y un valor,
#y devuelva la cantidad de veces que aparece ese valor en la lista

def ocurrencias(lista,valor):
    # IMPLEMENTAR
```

```
    return 0


l=[1,4,2,3,5,1,4,2,3,6,1,7,1,3,5,1,1,5,3,2]
v=2

print("La cantidad de ocurrencias es:")
print(ocurrencias(l,v))
#debe imprimir 3, la cantidad de veces que aparece el 2 en la lista
```

Clases

Python admite una forma limitada de herencia múltiple en clases. Las variables y métodos privados se pueden declarar (por convención, el lenguaje no lo impone) agregando un guión bajo (por ejemplo, _spam). También podemos vincular nombres arbitrarios a instancias de clase. A continuación se muestra un ejemplo:

In [ ]:

```
class Clase(object):
    comun = 10
    def __init__(self):
        self.myvariable = 3

    def myfunction(self, arg1, arg2):
        return self.myvariable

claseinstance = Clase()
print(" - Ejecutar funcion")
print(claseinstance.myfunction(1, 2))

claseinstance2 = Clase()
print(" Variable de clase, atributo comun entre las clase")
print(claseinstance.comun)
print(claseinstance2.comun)

print(" Cambiar atributo comun")
Clase.comun = 30
print(claseinstance.comun)
print(claseinstance2.comun)

print(" Cambiar el atributo comun solo de una instancia")
claseinstance.comun = 15
print(claseinstance.comun)
print(claseinstance2.comun)
```

Constructores de la clase

Para enviar paramatros al constructor de la clase se debe definir en el metodo `init`.

In [ ]:

```
class Persona(object):
    def __init__(self, nombre, apellido ):
        self.nombre = nombre
        self.apellido = apellido

    def obtenerNombres(self):
        return str(self.nombre) + " " + str(self.apellido)

persona1 = Persona("Diego", "Quisi")
persona2 = Persona("Juan", "Perez")

print(persona1.obtenerNombres())

print(persona2.nombre)
```

In [ ]:

```
# Ejercicio 7
```

```
# Generar un CRUD (Crear, Leer, Actualizar y Eliminar) de una agenda de telefono.

# IMPLEMENTAR

import os
def agenda():
    contactos = {}
    salir=True
    while(salir):

        print('Bienvenido A Mi Agenda\n')
        print(' 1.) Crear Contactos\n 2.) Leer Contacto\n 3.) Actualizar Contacto\n 4.) Eliminar C
ontacto\n 5.) Salir\n')

        opcion=input('Digite el numero de la opcion que desea ver: ')
        os.system('clear')
        if opcion == '1': # <----Crear los contactos
            nombre=input('Nombre: ')
            telefono=input('Telefono: ')
            contactos[(nombre)] = telefono
            print('Contacto agregado')
            print(contactos)
            os.system('clear')

        elif opcion == '2': # <----Leer los contactos
            for contacto in contactos:
                for numero in contactos:
                    print('Contacto / Numero')
                    print( numero,contactos[contacto])
                    os.system('clear')

        elif opcion == '3': # <------- Actualizar contacto
            contacto = input('Contacto a modificar: ')
            if contacto not in contactos:
                print ('El contacto no existe, agreguelo desde el menu')
                os.system('clear')

        elif opcion == '4': # <------- Eliminarr contacto
            eliminar=input('Contacto a eliminar: ')
            if eliminar not in contactos:
                print ("El contacto no existe")
            del(contactos[eliminar])
            print('Contacto',eliminar,'eliminado con exito')
            os.system('clear')

        elif opcion == '5': # <---- regresar al menu principal
            os.system('exit')
        else:
            print('Opcion no valida,\nElija una opcion del 1 al 5')
            os.system('clear')
agenda()
```

Excepciones

Las excepciones en pyhton permiten controlar a traves de la siguiente estructura:

In [ ]:

```
try:
    # Division by zero raises an exception
    10 / 0
except ZeroDivisionError:
    print("Oops, division invalida.")
else:
    # Si no existe ninguna excepcion no ocurre nada.
    pass
finally:
    # Se ejecuta exista o no la excepcion.
    print("Se ejecuta el bloque finally.")
```

In [ ]:

```
#Ejercicio 8
```

```
# Crear un metodo de validacion de cedula Ecuatoriana, en caso de que la cedula no sea validad lan
zar
# una excepcion, ademas de controlar que solo pueda ingresar digitos numericos por teclado.

def vcedula(texto):
    # sin ceros a la izquierda
    nocero = texto.strip("0")

    cedula = int(nocero,0)
    verificador = cedula%10
    numero = cedula//10

    # mientras tenga números
    suma = 0
    while (numero > 0):

        # posición impar
        posimpar = numero%10
        numero   = numero//10
        posimpar = 2*posimpar
        if (posimpar  > 9):
            posimpar = posimpar-9

        # posición par
        pospar = numero%10
        numero = numero//10

        suma = suma + posimpar + pospar

    decenasup = suma//10 + 1
    calculado = decenasup*10 - suma
    if (calculado  >= 10):
        calculado = calculado - 10

    if (calculado == verificador):
        validado = 1
    else:
        validado = 0
    return (validado)
num = input("Ingrese su cedula:")
if vcedula(num) == 0:
    print ("cedula incorecta")
else:
    print ("cedula corecta")


#IMPLEMENTAR
```

Importar

Las bibliotecas externas se usan con la palabra clave import [libname]. También puede usar desde [libname] import [funcname] para funciones individuales. A continuacion un un ejemplo:

In [ ]:

```
import pickle
mylist = ["This", "is", 4, 13327]
# Abre o crea un archivo binary.dat para guardar la informacion
myfile = open(r"binary.dat", "wb")
pickle.dump(mylist, myfile)
myfile.close()
print("archivo creado y almacenado el listado adjunto ")
```

In [ ]:

```
# Ejercicio 9
# Crear una aplicacion que me permita guardar en archivo y recuperar la informacion de nombres de
animales.

#IMPLEMENTAR
import pickle
```

```
class Animal:

    def __init__(self,nombre):
        self.nombre = nombre

    def __str__(self):
        return self.nombre

# Creamos la lista con los objetos
nombres = ["Pato","Perro","Gato"]
animales = []

for n in nombres:
    a = Animal(n)
    animales.append(a)

# Escribimos la lista en el fichero con pickle
import pickle
f = open('animales.pckl','wb')
pickle.dump(animales, f)
f.close()

# Leemos la lista del fichero con pickle
f = open('animales.pckl','rb')
animales = pickle.load(f)
f.close()

for a in animales:
    print(a)
```

Practica Integradora

Realizar una aplicacion para agendar contactos, en donde una Persona (nombre, apellido, cedula, fecha de nacimiento, direccion) puede tener varios Telefonos (numero, tipo de telefono (enum), codigo de area, operadora)., adicionalmente esta informacion se debe almacenar en un archivo para recuperarla. Los datos deben ser ingresados mediante consola, para ello se debe generar un menu interactivo que permita gestionar toda la informacion de la agenda (CRUD -> Personas y Telefonos), ademas de guardar y leer los datos del archivo. Incluir la validacion de la cedula y de datos de ingreso con excepciones.

In [ ]:

```
#IMPLEMENTAR AGENDA

import pickle
import os
class Contact:
    def __init__(self,name,email,phone):
        self.name=name
        self.email=email
        self.phone=phone

    def __str__(self):
        return "Name:{0}\nEmail address:{1}\nPhone:{2}".format(self.name,self.email,self.phone)

    def change_name(self,name):
        self.name=name

    def change_email(self,email):
        self.email=email

    def change_phone(self,phone):
        self.phone=phone

def add_contact():
    address_book_file=open("address_book_file","r")
    is_file_empty=os.path.getsize("address_book_file")==0
    if not is_file_empty:
        list_contacts=pickle.load(address_book_file)
    else:
        list_contacts=[]
    try:
        contact=get_contact_info_from_user()
        address_book_file=open("address_book_file","w")
        list_contacts.append(contact)
```

```
        pickle.dump(list_contacts,address_book_file)
        print ("Contact added")
    except KeyboardInterrupt:
        print ("Contact not added")
    except EOFError:
        print ("Contact not added")
    finally:
        address_book_file.close()

def get_contact_info_from_user():
    try:
        contact_name=input("Enter contact name\n")
        contact_email=input("Enter contact email\n")
        contact_phone=input("Enter contact phone number\n")
        contact=Contact(contact_name,contact_email,contact_phone)
        return contact
    except EOFError as e:
        #print "You entered end of file. Contact not added"
        raise e
    except KeyboardInterrupt as e:
        #print "Keyboard interrupt. Contact not added"
        raise e

def display_contacts():
    address_book_file=open("address_book_file","r")
    is_file_empty=os.path.getsize("address_book_file")==0
    if not is_file_empty:
        list_contacts=pickle.load(address_book_file)
        for each_contact in list_contacts:
            print (each_contact)
    else:
        print ("No contacts in address book")
        return
    address_book_file.close()

def search_contact():
    #search_name=input("Enter the name\n")
    address_book_file=open("address_book_file","r")
    is_file_empty=os.path.getsize("address_book_file")==0
    if not is_file_empty:
        search_name=input("Enter the name\n")
        is_contact_found=False
        list_contacts=pickle.load(address_book_file)
        for each_contact in list_contacts:
            contact_name=each_contact.name
            search_name=search_name.lower()
            contact_name=contact_name.lower()
            if(contact_name==search_name):
                print (each_contact)
                is_contact_found=True
                break
        if not is_contact_found:
            print ("No contact found with the provided search name")
    else:
        print ("Address book empty. No contact to search")
    address_book_file.close()

def delete_contact():
    #name=input("Enter the name to be deleted\n")
    address_book_file=open("address_book_file","r")
    is_file_empty=os.path.getsize("address_book_file")==0
    if not is_file_empty:
        name=input("Enter the name to be deleted\n")
        list_contacts=pickle.load(address_book_file)
        is_contact_deleted=False
        for i in range(0,len(list_contacts)):
            each_contact=list_contacts[i]
            if each_contact.name==name:
                del list_contacts[i]
                is_contact_deleted=True
                print("Contact deleted")
                address_book_file=open("address_book_file","w")
                if(len(list_contacts)==0):
                    address_book_file.write("")
                else:
                    pickle.dump(list_contacts,address_book_file)
                break
```

```
        if not is_contact_deleted:
            print ("No contact with this name found")

    else:
        print ("Address book empty. No contact to delete")
    address_book_file.close()

def modify_contact():
    address_book_file=open("address_book_file","r")
    is_file_empty=os.path.getsize("address_book_file")==0
    if not is_file_empty:
        name=input("Enter the name of the contact to be modified\n")
        list_contacts=pickle.load(address_book_file)
        is_contact_modified=False
        for each_contact in list_contacts:
            if each_contact.name==name:
                do_modification(each_contact)
                address_book_file=open("address_book_file","w")
                pickle.dump(list_contacts,address_book_file)
                is_contact_modified=True
                print ("Contact modified")
                break
        if not is_contact_modified:
            print ("No contact with this name found")
    else:
        print ("Address book empty. No contact to delete")
    address_book_file.close()

def do_modification(contact):
    try:
        while True:
            print ("Enter 1 to modify email and 2 to modify address and 3 to quit without modifying
")
            choice=input()
            if(choice=="1"):
                new_email=input("Enter new email address\n")
                contact.change_email(new_email)
                break
            elif(choice=="2"):
                new_phone=input("Enter new phone number\n")
                contact.change_phone(new_phone)
                break
            else:
                print ("Incorrect choice")
                break
    except EOFError:
        print ("EOF Error occurred")
    except KeyboardInterrupt:
        print ("KeyboardInterrupt occurred")

print ("Enter 'a' to add a contact, 'b' to browse through contacts, 'd' to delete a contact, 'm' t
o modify a contact, 's' to search for contact and 'q' to quit")
while True:
    choice=input("Enter your choice\n")
    if choice == 'q':
        break
    elif(choice=='a'):
        add_contact()
    elif(choice=='b'):
        display_contacts()
    elif(choice=='d'):
        delete_contact()
    elif(choice=='m'):
        modify_contact()
    elif(choice=='s'):
        search_contact()
    else:
        print ("Incorrect choice. Need to enter the choice again")
```

```
Enter 'a' to add a contact, 'b' to browse through contacts, 'd' to delete a contact, 'm' to modify
a contact, 's' to search for contact and 'q' to quit
```

Otros tutoriales

Este tutorial corto intenta darte los elementos mínimos de python para poder trabajar, para algunas temas que no se trataron por favor ingresar al siguiente link Python 2 También para complementar este recurso con el curso online de Python de CodeAcademy o este libro de python.