Guía Laboratorio 1.1
Procesamiento Digital de Señales

Alejandro Escobar, Cristian Ríos, Paula Pérez

2024-1

1. Introducción

En este laboratorio se pretende que el estudiante repase conceptos básicos de programación y aprenda las bases de programación en Python que se usarán durante el curso.

Nota: Los comentarios dentro de los códigos fuentes en esta guía no llevan tilde. Por defecto, Python contempla que estamos utilizando caracteres ASCII de 7 bits.

2. Anaconda

La instalación en Windows se hace con un instalador y no tiene problemas. El programa *jupyter-notebook* se abre desde el Anaconda Navigator.
Para la instalación en Linux siga los siguientes pasos:

1. Descargue Anaconda para Python 3.9 del siguiente enlace:
   `https://www.anaconda.com/products/individual`
   En los computadores del laboratorio ya se encuentra instalado en Linux.

2. Abra la carpeta donde descargó el instalador y dé clic derecho en un espacio en blanco, seleccione *Abrir en Terminal* (o abra una terminal y con el comando cd vaya a la carpeta).

3. Instale Anaconda con el siguiente comando:

   ```
   bash Anaconda...
   ```

   La licencia se lee oprimiendo *Espacio* hasta que se pida escribir "*yes*". Procure instalarlo en una ubicación que no tenga tildes o caracteres extraños en su ruta.

4. Al final de la instalación el programa pregunta algo más, indique sí con una "*y*" y presione *Enter*.

5. Cierre la Terminal y abra una nueva, ejecute el siguiente comando:

   ```
   jupyter-notebook
   ```

3. Python básico

3.1. Variables

Python es un lenguaje tipado de manera dinámica, es decir, que la misma variable puede tomar diferentes tipos (entero, flotante, caractéres, etc.) en el mismo programa. A continuación se muestra un ejemplo del manejo de algunas variables.

```
# Una variable puede contener una cadena de caracteres
a='Spam'
print(a)

# Se pueden concatenar cadenas con el operador '+'
b=' with eggs'
print(a+b)

#%%
# Ahora la misma variable de antes puede contener un entero
a=42
print(a)

# Naturalmente, se puede operar entre enteros
b=8
print(a+b)
```

Para mostrar información se pueden imprimir directamente las variables o se puede utilizar la función *format*.

```
c='The answer to the Ultimate Question of Life, the Universe, and Everything is:'
print(c)
print(a)

# Se pueden concatenar resultados utilizando coma (,)

print(c,a)

# De una manera mas 'elegante' con la instruccion "format"
c='The answer to the Ultimate Question of Life, the Universe, and Everything is: {}!'
print(c.format(a))
d=2
e='Este es'
print('{} otro ejemplo de format con {} variables'.format(e,d))
```

Para mostrar información se pueden imprimir directamente las variables o se puede utilizar la función *format*.

```
# En Python 3, la division entre enteros produce un numero flotante
a=2
b=3
print(b/a)

# Para lograr un entero se debe utilizar el operador //
print(b//a)

# El operador "**" permite hacer potenciacion
print(a**b)
```

Los arreglos sirven para guardar datos del mismo tipo, es decir, todos sus elementos deben ser enteros, cadenas de caracteres, etc.

```
arreglo_num=[3, 1, 7, 6]
arreglo_str=['primer', 'segundo', 'tercer', 'cuarto']
print(arreglo_num)
print("El tercer elemento es: {}".format(arreglo_num[2]))
print(arreglo_str)

for i,s in enumerate(arreglo_str): # Recorrer los arreglos, mas de esto despues
    print("El {} elemento es: {}".format(s,arreglo_num[i]))
```

Las listas permiten guardar diferentes tipos de datos bajo el mismo nombre.

```
lista=[3, 'patata', 7, 2.5, 'spam']
for l in lista: # Recorrer la lista
    if isinstance(l, str): # Revisar si es una cadena de caracteres
        print(l)
    else:
```

```
        print(1/2)
```

Un diccionario permite almacenar como un conjunto de clave y valor sin orden.

```
diccionario={'rojo':'red', 'azul':'blue', 'cien':100, 'dos':2}
print(diccionario['rojo'])
print(diccionario['cien'])
print(diccionario['dos'])
```

3.2. Instrucciones de control

Las principales instrucciones en Python son:

- ***if***: Ejecutar cierto código de acuerdo a si se cumple una condición.
- ***for***: Ciclos con un número específico de iteraciones.
- ***while***: Repetir código hasta cumplir una condición.

Python no ofrece otras instrucciones como *switch* o el ciclo *do-while*.

```
a=2
b=3
if a>b:
    print('a es mayor que b')
elif a==b:
    print('a es igual a b')
else:
    print('a es menor que b')

# Se pueden hacer comparaciones entre diferentes tipos sin error
a=42
if a == 'The answer to the Ultimate Question of Life, the Universe, and Everything':
    print('Spam!')
else:
    print("It does not work :(")

# Se puede buscar una cadena dentro de otra
a='studying' # Intente con: 'assassin'
b='dying' # Y 'sin'
if a.find(b):
    print("You can't spell {} without {}".format(a,b))
```

Python distingue entre ciclos *for* y ciclos *while*, definiéndolos según el "libro". Es decir, los ciclos *for* son para un número conocido de iteraciones, mientras que los ciclos *while* se utilizan cuando no se conoce de antemano el número de iteraciones, sino que se itera hasta cumplir cierta condición.

Los ciclos *for* en Python siempre se hacen sobre secuencias almacenadas en arreglos o listas, como se muestra en los siguientes ejemplos:

```
#%%
# Los ciclos for son para secuencias almacenadas en arreglos o listas
str_seq=['Esto', 'es', 'una', 'secuencia', 'de', 'cadenas', 'de', 'caracteres']
for s in str_seq:
    print(s)

# Pueden ser secuencias de numeros
num_seq=[3, 8, 2, 5, 32, 90]
for num in num_seq:
    print("El numero es: {}".format(num))

# Si necesito el indice utilizo enumerate
for i,num in enumerate(num_seq):
    print("El numero en la posicion {} es: {}".format(i,num))
```

```
# Pueden ser tuplas
str_seq=[[2, 'Esto'], [5, 'es'], [7, 'una'], [1, 'secuencia'], [3, 'de'], [9, 'tuplas']]
for s1,s2 in str_seq:
    print("{} : {}".format(s1,s2))
```

Los ciclos *while* se repiten hasta que se cumple una condición.

```
a=32
while a!=42:
    a+=2
    print('Keep searching!')
c='The answer to the Ultimate Question of Life, the Universe, and Everything is: {}!'
print(c.format(a))
```

3.3. Funciones

Como en todos los lenguajes de programación, en Python el programador puede definir sus propias funciones. Estas se declaran con la palabra def y se definen de la siguiente manera.

```
def funcion(arg1, arg2):
    print("El primer argumento es: {}".format(arg1))
    print("El segundo argumento es: {}".format(arg2))
    return arg1*2, arg2*3

a=1
b=2

c,d=funcion(a,b)

print("El primer valor retornado es: {}".format(c))
print("El segundo valor retornado es: {}".format(d))
```

3.4. Librerías propias

Si tenemos alguna función que queremos utilizar en varios programas podemos aprovechar el uso de librerías para evitar tener que copiar y pegar en varios códigos.
Definamos dentro de un archivo llamado *MiLibreria.py* las siguientes dos funciones.

```
def funcion1(arg1):
    print("Esta funcion multiplica por 2")
    return 2*arg1

def funcion2(arg1):
    print("Esta funcion multiplica por 3")
    return 3*arg1
```

Y ahora podemos utilizar estas funciones de la siguiente manera:

```
import MiLibreria

a=MiLibreria.funcion1(1)
print(a)
b=MiLibreria.funcion2(2)
print(b)
```

4. Ejercicios en Python

1. Escriba un programa que pregunte la edad del usuario, y determine si es mayor de edad o no.
2. Escriba un programa que pregunte la edad del usuario, luego muestre la letra de la canción *Cumpleaños Feliz* y cuando llegue a la parte de la canción donde se cuenta hasta la edad utilice un ciclo para contar. Practique utilizando un ciclo *for* y con un ciclo *while*.
3. Escriba un programa que solicite iterativamente un caracter ingresado por consola (puede usar un ciclo *while* para esto). El programa debe identificar si el caracter ingresado es mayúscula, minúscula, o ninguna de las dos; además imprima un mensaje según el caso. El programa debe finalizar cuando se ingrese un caracter especial, por ejemplo "1".

Tip: Busque información sobre como utilizar las funciones *input* y *raw_input*, además considere el uso de la función *ord()* de python para el punto 3.

5. Manejo básico de señales en Python

Información adicional que será útil para este y los próximos laboratorios:
`https://github.com/kuleshov/cs228-material/blob/master/tutorials/python/cs228-python-tutorial.ipynb`

5.1. Arreglos y Matrices

1. Manejo de arreglos y matrices

```
# manejo de arreglos

import numpy as np    # Libreria estandar para manejo de datos y senales en Python

a = np.zeros((2,2))   # Crea un arreglo tipo matriz de ceros
print(a)              # Prints "[[ 0.  0.]
                      #          [ 0.  0.]]"
b = np.ones((1,2))    # Crea un arreglo tipo vector fila de unos
print(b)              # Prints "[[ 1.  1.]]"

c = np.full((2,2), 7) # Crea un arreglo tipo matriz de un valor constante
print(c)              # Prints "[[ 7.  7.]
                      #          [ 7.  7.]]"
c = 7*np.ones((2,2))  # Otra forma
print(c)

d = np.eye(2)         # Crea un arreglo con la matriz identidad
print(d)             # Prints "[[ 1.  0.]
                      #          [ 0.  1.]]"
e = np.random.random((2,2)) # Crea un arreglo tipo matriz de numeros aleatorios entre 0 y 1
print(e)             # print "[[ 0.91940167  0.08143941]
                      #        [ 0.68744134  0.87236687]]"
print(e[0,1])        # Seleccionar un elemento
print(e[0,:])        # Seleccionar la primera fila
print(e[:,1])        # Seleccionar la segunda columna

xrang=np.arange(20)   # Crea un arreglo de numeros consecutivos entre 0 y 19
print(xrang)         # print "[0 1 2 3 4 5... 19]"

# Para seleccionar partes del arreglo use los siguientes comandos

print(xrang[0:5])     # print "[0 1 2 3 4]"
print(xrang[10])      # print [10]

A=np.random.random(100)
media=np.mean(A)      # calcula la media de un arreglo A
maxA=np.max(A)        # calcula el maximo valor de un arreglo
minA=np.min(A)        # calcula el minimo valor de un arreglo
print(media, maxA, minA)
```

2. Con la información anterior, desarrolle un script que calcule la media, la desviación estándar, el máximo, el mínimo para el arreglo:

```
datos=np.array([2, 4, 16, 8, 19, 35, 53, 100, 24, 35, 44, 26, 145, 1, 1, 1, 0])
```

5.2. Manejo del notebook

Los notebooks de jupyter serán los informes que se entregarán de cada práctica de laboratorio. Tener en cuenta las siguientes consideraciones.

1. Existen diferentes tipos de celdas. Las más comunes son:

   - **Code:** En estas se escribe el código que se va a ejecutar
   - **Markdown:** En estas se puede escribir texto, conclusiones, y explicaciones.
     Cree una nueva celda **Markdown**, y escriba el siguiente codigo.

```
# Titulo I
## Titulo II
### Titulo III

Texto normal

Se pueden agregar ecuaciones como si fuera codigo de LaTex

$$E=\frac{1}{N}\sum_{i=1}^N x[i]^2$$

Se pueden incrustar imagenes en el notebook como si fuera codigo HTML

<img src="http://cdn.memegenerator.es/imagenes/memes/full/4/10/4107176.jpg" width="500" ...
    align="middle">

Se pueden agregar enlaces a paginas web

[Introduccion a Jupyter](http://www.emilkhatib.es/introduccion-a-jupyter-ipython-notebook/)
```

5.3. Manejo de señales

1. El siguiente ejemplo genera una señal sinusoidal con una frecuencia de 1 Hz, una duración de 2 s y una frecuencia de muestreo de fs=5 Hz.

```
import numpy as np
import matplotlib.pyplot as plt # libreria usada para graficas

# para que la grafica se incruste en el notebook
%matplotlib inline

f=1.0  # Frecuencia de la senal
fs=5.0 # Frecuencia de muestreo
t=np.arange(0, 2.0, 1.0/fs) # Vector de tiempo
x = np.sin(2*np.pi*f*t)
plt.plot(t,x)
plt.xlabel('Time',fontsize=18)
plt.ylabel('Amplitude',fontsize=18)
plt.show()
```

   ¿Cómo se ve la figura?

2. Cambie la frecuencia de muestreo de la señal por 20 Hz, y ejecute nuevamente el script, ¿Qué cambios se observan?

3. También se pueden cargar señales de distintas fuentes de información, por ejemplo archivos de Excel. Ver el siguiente ejemplo que analiza la tasa de desempleo en Colombia desde 2002 hasta 2017.

```
import pandas as pd # libreria para el manejo de datos

# Leer datos
TD = pd.read_excel("TD.xlsx")

# Cambio de formato de los datos
TD = pd.DataFrame(TD)

# Extraigo una de las columnas del archivo de excel
TDC = TD["TD"]
index = TD["Index"]

#grafica de la tasa de desempleo
plt.plot(index, TDC)
plt.title("Tasa de desempleo en Colombia")
```

¿Qué se puede concluir?

Pueden ver en este enlace información adicional.
`https://github.com/neuraldevs/ML-ND-CD/blob/master/Regression/Regression.ipynb`

4. Se pueden cargar también señales de audio, tal como en el siguiente ejemplo. Primero Descargue la señal de audio de su preferencia del siguiente enlace, cópiela a la misma carpeta del *Notebook* y renómbrela como *senal.wav*.

   `https://ccrma.stanford.edu/~jos/pasp/Sound_Examples.html` en formato **.wav*

```
from scipy.io.wavfile import read # libreria para lectura de archivos de audio
from IPython.display import Audio # para escuchar la senal

file_audio=('senal.wav') # Ruta del archivo con la senal
fs, x=read(file_audio) # Cargar el archivo
x=x/float(max(abs(x))) # escala la amplitud de la senal
t=np.arange(0, float(len(x))/fs, 1.0/fs) # Vector de tiempo
plt.plot(t,x) # Dibujar la grafica

# Los siguientes dos comandos dibujan las etiquetas de los ejes
plt.xlabel('Time',fontsize=18) # Etiqueta eje X
plt.ylabel('Amplitude',fontsize=18) # Etiqueta eje Y
plt.show() # Mostrar la grafica
Audio(x, rate=fs) # para escuchar la senal, si se desea
```

5. Invierta la señal de audio en el dominio temporal, es decir, la última muestra pasa a ser la primera, la penúltima se vuelve la segunda, ... ¿Cómo se realizaría el procedimiento?. Escuche la señal invertida, ¿Qué observa?

6. Funciones: Calcule el valor DC y la energía de la señal de audio anterior

```
def logEnergy(sig):      # Definir la funcion
    sig2=np.square(sig)  # Elevar al cuadrado las muestras de la senal
    sumsig2=np.sum(sig2) # Sumatoria
    logE=10*np.log10(sumsig2) # Convertir a dB
    dc=np.mean(sig)      # Promedio
    return logE, dc

# Ahora se usa la funcion anterior para calcular las medidas de la senal
Energy, DCvalue = logEnergy(x)
print('Energia='+str(Energy))
print('Valor DC='+str(DCvalue))
```

5.4. Procedimiento

Diseñar una función en python que tenga como entrada un archivo de audio, y realice el siguiente procedimiento para realizar un detector de silencio.

- Cargue y normalice la señal de audio

- Divida la señal en segmentos de 10 ms de duración (en este punto se debe realizar un ciclo for)
- A cada segmento de 10 ms debe calcularle la energía.
- Cada valor de energía debe ser almacenado en una lista.
- Cuando la energía sea inferior a un umbral determinado, el segmento debe ser eliminado de la señal.
- La función debe retornar la señal luego de eliminarle los silencios.

Finalmente, grafique la señal de entrada y la señal de salida usando subplots. Escuche ambas señales y concluya.