UNIVERSIDAD NACIONAL AUTÓNOMA DE MÉXICO

FACULTAD DE PSICOLOGÍA

CURSO DE INTRODUCCIÓN A LA PROGRAMACIÓN EN *PYTHON*

JUNIO 2017

Edgar de Jesús Vázquez Silva
Laboratorio 25, Facultad de Psicología

Índice

1. Introducción

Este documento tiene la finalidad de ayudar a los alumnos de psicología a entender la estructura de un programa en Python, crear un programa básico, además de conocer algunas librerías útiles, por ejemplo: generación de gráficos.

2. Conociendo Python

Como estudiantes, profesores o curiosos de la programación hemos de buscar un lenguaje de programción que nos sea útil según nuestras necesidades. Es importante mencionar que existen muchos, y muy diversos, lenguajes de programación: R, Ruby, Java, Python, C, C++, por mencionar algunos. Cada uno de ellos ofrece características que lo hacen destacar según las necesidades del desarrollador.

Para comenzar a programar en Python necesitaremos una interfaz que nos facilite la tarea de generar Çódigos"(programas computacionales), lo más recomendable para las personas que no han tenido un acercamiento a un lenguaje de programación es utilizar una *interfaz gráfica conocida como IDLE.* Otra opción es utilizar un editor de texto como SublimeText, NotePad++ ó Atom, y ejecutarlo desde la consola. Se recomienda al lector consultar la guía de instalación del IDLE Spyder, desarrollado por los alumnos del LAB 25. `https://github.com/Lab25UNAM/PAPIME2016/blob/master/guia_Instalacion.pdf`

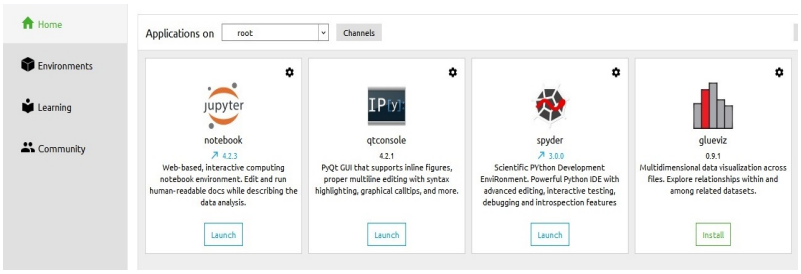
En los siguientes ejemplos, las entradas y salidas son distinguidas por la presencia o ausencia de los prompts >>>: para reproducir los ejemplos, debes escribir todo lo que esté después del prompt, cuando este aparezca; las líneas que no comiencen con el prompt son las salidas del intérprete.

Una vez instalado la suite de desarrollo Anaconda, buscaremos el entorno de desarrollo *Spyder.* Con esto estaremos listos para comenzar a desarrollar programas en lenguaje de programación Python.

2.1. Escribiendo un hola mundo

Una práctica muy conocida en el mundo de programación es comenzar a escribir un programa de bienvenida al lenguaje de programación. Para ello seguiremos estos sencillos pasos:

- Buscar Anaconda en en menú inicio del sistema.
- Abrir el IDE Spyder.
- Una vez en editor de Spyder, escribir el siguiente comando.




```
1        print "Hola mundo...."
2
```

Con lo cual observaremos en la terminal del editor el mensaje anteriormente escrito.

3. Variables

Una de las características más poderosas en un lenguaje de programación es la capacidad de manipular variables. Se entiende como variable el nombre dado a un valor.

La sentencia de asignación crea nuevas variables y les da valores:

```
mensaje = "Hola mundo.... :)"
n = 17
pi = 3.14159

```

Este ejemplo hace tres asignaciones. La primera asigna la cadena *"Hola mundo.... :)"* a una nueva variable denominada mensaje. La segunda le asigna el entero 17 a n, y la tercera le da el valor decimal 3.14159 a pi.

3.1. Tipos de variables

Hasta este punto ya hemos declarado algunas variables; sin embargo es importante conocer que tipo de valores podemos asignar a estas variables. Dentro de los tipos de datos principales que podemos utilizar en Python estan:

- Boleanos (bool): Son tipos de datos que solo pueden tomar dos valores True o False (Verdader o Falso). Este tipo de variables nos pueden ayudar a guardar información que responda a una pregunta de afirmación. Por ejemplo, ¿Una persona es mayor de edad?, ¿Aún hay leche en el refrigerador? ¿Una paloma ha presionado el botón azul?, etc.
- Enteros (int): Se refiere a los tipos de datos que podemos representar con un numero entero. Por Ejemplo el numero de hermanos que una persona puede tener.
- Flotantes (float): Los tipos de datos flotantes se refieren a valores numéricos con punto decimal, Por ejemplo para almacenar el valor de la estatura de una persona utilizaremos una varible de tipo flotante.
- Cadenas (string): Son tipos de datos que se refieren a un conjunto de caracteres. Este tipo de datos son muy utilizados para manejo de textos, por ejemplo despliegue de mensajes o información.

- Arreglos (Arrays): Este tipo de datos en realidad es una extensión de los anteriores, y se refiere a la forma en que organizamos los datos. Un arreglo es un acomodo especial de datos de forma contigua, a la cual accedemos mediante un indice. Por ejemplo, si queremos guardar las calificaciones de 25 alumnos es más conveniente hacerlo en un arreglo de flotantes llamado çalificaciones.[en] lugar de declarar 25 variables de tipo flotante.

Muchos de los ejemplos de este manual, incluso aquellos ingresados en el prompt, incluyen comentarios. Los comentarios en Python comienzan con el carácter numeral, # , y se extienden hasta el final físico de la línea. Un comentario quizás aparezca al comienzo de la línea o seguidos de espacios blancos o código, pero sin una cadena de caracteres. Ya que los comentarios son para aclarar código y no son interpretados por Python, pueden omitirse cuando se escriben ejemplos.

```
1 # Ejemplo de tipos de datos y la declaracion de variables
2 # Este es un comentario, debe comenzar con el simbolo #
3
4 paloma_boton_azul = True    # Variable tipo booleana
5 numero_mascotas = 3         # Variable de tipo entero
6 promedio = 7.56             # Variable de tipo flotante
7 Saludo = "Hola humano... :)"# Variable de tipo cadena
8
9
10 #Variable de tipo arreglo
11 calificaciones = [8.7, 9.7, 8.9, 8.1, 7.5, 6.8]
12
```

4. Estructuras de control condicionales

Una sentencia de control condicional es una estructura de programación que nos permite evaluar si una expresión es verdadera. Este tipo sentencias se emplean para responder una pregunta y tomar una desición en caso que esta sea verdadera o falsa.

4.1. if-else

La más simples de estas estructuras es la sentencia *if*. Esta sentencia evalua una condicion presentada.

```
if expresion_1 == expresion_2:
    hacerAlgo()
```

Para hablar de estructuras de control de flujo en Python, es imprescindible primero, hablar de identación. **¿Qué es la identación?** En un lenguaje informático, la identación es lo que la sangría al lenguaje humano escrito (a nivel formal).

Así como para el lenguaje formal, cuando uno redacta una carta, debe respetar ciertas sangrías, los lenguajes informáticos, requieren una identación. No todos los lenguajes de programación, pero en el caso de Python, la identación es obligatoria, ya que de ella, dependerá su estructura.

Una identación de 4 (cuatro) espacios en blanco, indicará que las instrucciones identadas, forman parte de una misma estructura de control.

```
if expresion_1 == expresion_2:
    leerVariable          # Esto pasara si la comparacion
    compararVariable      # de varibles es verdadera
    imprimirMensaje
    multiplicarVariables

imprimirMensajeFinal      # Esto ya no forma parte de la
                          # estructura   if
```

En este ejemplo podemos observar que se verifica que la expresión_1 sea igual que la expresión_2. En caso de ser afirmativa la respuesta se ejecutarán las instrucciones indicadas por la **identación**. *Es importante señalar que en este primer ejemplo, cuando las expresiones NO son iguales no se toma ninguna acción.* Esta observación es importante; pues esta primera estructura

nos servirá cuando queremos evaluar una condicion y solo nos importen tomar acción en ese caso.

A continuación se enuncia un ejemplo en el cual nos interesa realizar una acción cuando las sentencias sean verdaderas y otro acción diferente cuando la sentencia NO sea verdadera.

La estrcutura if-else por su parte ofrece una alternativa de acción en caso que la sentecia a comparar sea falsa. Esta estructura puede declararse como se muestra a continuación:

```
if expresion_1 == expresion_2:
    leerVariable          # Esto pasara si la comparacion
    compararVariable      # de varibles es verdadera
    imprimirMensaje
    multiplicarVariables
else
    imprimirMensaje       # Esto pasara si la comparacion
    terminarPrograma      # es falsa

imprimirMensajeFinal      # Esto ya no forma parte de la
                          # estructura   if
```

Como observamos en el ejemplo 2, primero evaluaremos que la expresión_1 y la expresion_2 sean iguales, si ocurre el caso en que ambos tengan el mismo valor se ejecutará la secuencia de pasos número_1; en el caso contrario cuado la expresión_1 y la expresión_2 **NO** sean iguales se ejecutará la secuencia de pasos número_2.

Con la finalidad de ser más claro con el lector a continuación se mencionan algunos ejemplos.

```
# En este primer caso solo nos interesa realizar
# una accion cuando la sentencia sea verdadera
# Por lo tanto no lleva condicional else

if velocidadAutomovil > 90:
    multarCondutor
```

En este primer caso no es relevante saber si el conductor va a una velocidad menor que la límite.

```
# En este segundo caso nos interesa realizar
# una accion cuando la sentencia sea verdadera
```

```
# y otra accion cuando sea diferente

if velocidadAutomovil > 90:
    if edadConductor > 18:
        multarConductor
    else
        llamarAlosPadres
```

En este ejemplo, nos interesa saber dos cosas: si el automóvil va a una velocidad mayor que la límite (en este caso 90 km/h) y si el conductor es menor o mayor de edad.

Hasta ahora ya hemos visto como preguntar si dos expresiones son iguales; sin embargo hay más comparaciones que podemos realizar. A continuación se enlistan los comparadores más comunes en Python.

Simbolo	**Significado**	**Ejemplo**	**Resultado**
==	igual que	89 == 7	Falso
!=	diferente	verde != rojo	Verdadero
<	menor que	12.0 < 12.5	Verdadero
>	mayor que	8 > 10	Falso
<=	menor o igual que	12 <= 12	Verdadero
>=	mayor o igual que	15 >= 20	Falso

5. Ciclos condicionales

A diferencia de las estructuras de control condicionales, las iterativas (también llamadas cíclicas o bucles), nos permiten *ejecutar un mismo código, de manera repetida*, mientras se cumpla una condición.

En Python se dispone de dos estructuras cíclicas:

- El bucle while
- El bucle for

Las veremos en detalle a continuación.

5.1. Ciclo while

Este bucle, se encarga de ejecutar una misma acción **"mientras que"** una determinada condición se cumpla.

Ejemplo: *Mientras que año sea menor o igual a 2017, imprimir la frase Ïnforme del año _anio_"*

```
#Declaramos e inicializamos una variable
anio = 2000

while anio <= 2017:
  print "Informe del ciclo ", str(anio)
  anio = anio+1
```

Podemos notar que el ciclo while repetirá la impresión de un mensaje mientras que el valor de la variable anio sea menor o igual que 2017; además de realizar esta acción realizamos una acción más, el incremento de la variable en una unidad. **¿Que pasrá si no se incrementa esta variable?** la variable anio siempre tendrá un valor de 2000 y nunca saldrá del ciclo *while*. El ejemplo anterior tendrá una salida similar a esta:

```
Informe del ciclo 2000
Informe del ciclo 2001
Informe del ciclo 2002
Informe del ciclo 2003
Informe del ciclo 2004
.
.
```

```
8
9 .
10 Informe del ciclo 2017
```

Pero, ¿Que sucede si el ciclo while es infinito o demasiado largo? Por ejemplo deseamos realizar un programa que pregunte al usuario su nombre y si la persona se llama "Eduardo" imprimiremos un mensaje de felicitación. Para resolver situaciones de este tipo se emplea la instrucción "*break*".

```
1 while True:
2   nombre = raw_input("Indica tu nombre:  ")
3   if nombre == "Eduardo" :
4     break
5
6 print "Hola Eduardo, bienvenido al sistema..."
```

En este caso el ciclo while se repetirá infinitamente. El ciclo consiste en preguntar el nombre de una persona y lo guarda en una variable llamada "*nombre*", posteriormente verifica si el valor de la variable es igual a "*Eduardo*", si la respuesta es afirmativa va a instrucción *break* y sale del ciclo while. En caso que el nombre no sea igual a "*Eduardo*" el programa seguirá ejecutando el ciclo while, preguntando infinitamente.

5.2. ciclo for

El ciclo for en el lenguaje Python tiene cierta ventaja y diferencia con otros lenguajes. Podemos pensar que el ciclo For se define utilizando contadores y rangos en los cuales se ejecutaría el código del for.

A continuación la sintaxis de For en Python.

La sintaxis es la siguiente:

```
1   for iterador in secuencia
2       #codigo a ejecutar
```

Esto quiere decir que cuando usamos la sentencia For, tenemos la capacidad de recorrer una secuencia por medio de "iteraciones", una secuencia como una lista o una simple cadena de texto, veamos un ejemplo para comprender mejor.

Si quisiéramos declarar una cadena de texto y recorrer cada uno de sus caracteres, podemos usar la sentencia For para ello.

Este programa recorrera cada letra de la cadena de texto “Hola!” y la imprimira en pantalla.

```
#!/usr/bin/python

for letra in 'Hola!':
  print 'Estamos en la letra :', letra


Este seria el resultado:
Estamos en la letra : H
Estamos en la letra : o
Estamos en la letra : l
Estamos en la letra : a
Estamos en la letra : !

```

5.2.1. Iterar utilizando Indices (listas)

También es posible hacer iteraciones con For utilizando indices de listas. Esto quiere decir que la variable iterada tendra el valor de un indice, algo así como un contador común y corriente.

Ejemplo:

```
#!/usr/bin/python
autos = ['mercedez','BMW','Toyota']

for indice in range(len(autos)) #range define un rango que es el
    tamano de la lista
       print 'El auto es un ',autos[indice]



En este ejemplo iteramos una lista de autos y  los accedemos
    utilizando el indice de la lista, el resultado seria:

El auto es un mercedez
El auto es un BMW
El auto es un Toyota


```

```
16
17 La equivalencia sin usar el indice seria la siguiente:
18 #!/usr/bin/python
19
20 autos = ['mercedez','BMW','Toyota']
21
22 for auto in autos
23   print 'El auto es un ',auto
24
```

En cuyo caso la variable “auto” esta iterando la secuencia de la lista de autos, y toma el valor de cada uno de sus elementos dentro de esta lista.

6. Definición de funciones

En Python, la definición de funciones se realiza mediante la instrucción **def** más un nombre de función descriptivo para el cuál, aplican las mismas reglas que para el nombre de las variables seguido de paréntesis de apertura y cierre. Como toda estructura de control en Python, la definición de la función finaliza con dos puntos (:) y el algoritmo que la compone, irá identado con 4 espacios:

```
def mi_funcion():
    # aqui el algoritmo
```

Una función, no es ejecutada hasta tanto no sea invocada. Para invocar una función, simplemente se la llama por su nombre:

```
def mi_funcion():
    print "Hola Mundo"

funcion()
```

Cuando una función, haga un retorno de datos, éstos, pueden ser asignados a una variable:

```
def funcion():
    return "Hola Mundo"

frase = funcion()
print frase
```

6.0.2. Sobre los parámetros

Un parámetro es un valor que la función espera recibir cuando sea llamada (invocada), a fin de ejecutar acciones en base al mismo. Una función puede esperar uno o más parámetros (que irán separados por una coma) o ninguno.

```
def mi_funcion(nombre, apellido):
    # algoritmo
```

Los parámetros, se indican entre los paréntesis, a modo de variables, a fin de poder utilizarlos como tales, dentro de la misma función.

Los parámetros que una función espera, serán utilizados por ésta, dentro de su algoritmo, a modo de variables de ámbito local. Es decir, que los parámetros serán variables locales, a las cuáles solo la función podrá acceder:

```
def mi_funcion(nombre, apellido):
    nombre_completo = nombre, apellido
    print nombre_completo
```

En Python, también es posible, asignar valores por defecto a los parámetros de las funciones. Esto significa, que la función podrá ser llamada con menos argumentos de los que espera:

```
def saludar(nombre, mensaje='Hola'):
    print mensaje, nombre

saludar('Pepe Grillo') # Imprime: Hola Pepe Grillo
```

7. Librerías

7.1. numpy

NumPy es una extensión de Python, que le agrega mayor soporte para vectores y matrices, constituyendo una biblioteca de funciones matemáticas de alto nivel para operar con esos vectores o matrices. El objeto principal de NumPy es el array multidimensional homogéneo. Es una tabla de elementos (habitualmente números), todos del mismo tipo, indexados por una tupla de enteros positivos. En NumPy las dimensiones se llaman ejes. El número de ejes es el rango.

Hay muchas formas de crear un array NumPy. Por ejemplo:

```
# Esta linea es muy importante, con ella importamos la libreria y todos sus
# metodos para trabajar con ella.
import numpy as np

a = np.array([1, 2, 3, 4])
b = np.array([(1.5, 2, 3), (4, 5, 6)])
c = np.zeros( (3, 4) )
d = np.arange( 1, 10, 2 )
print a
print b
print c
print d

[1 2 3 4]
[[ 1.5  2.   3. ]
 [ 4.   5.   6. ]]
[[ 0.  0.  0.  0.]
 [ 0.  0.  0.  0.]
 [ 0.  0.  0.  0.]]
[1 3 5 7 9]
```

Los índices comienzan en cero. El índice -1 da el último elemento del array.

```
print a[0], b[0,0], b[0][0]
print d[-1]



1 1.5 1.5
9
```

ndim: número de dimensiones del array

shape: número de elementos en cada dimensión

```
print 'dim = ', b.ndim, '; shape = ', b.shape

dim =  2 ; shape =  (2, 3)
```

Usaremos principalmente los tipos estandar int* y float* (además de otros tipos estandar como boolean):

```
print a.dtype
print b.dtype

int64
float64
```

7.1.1. Operaciones básicas

Los operadores aritméticos se aplican en los arrays elemento a elemento. ¡En las listas Python funcionan de forma diferente!

```
a = np.array([1, 2, 3, 4])
b = np.array([(1.5, 2, 3, PI)])
print a+b
a1 = [1, 2, 3, 4]
b1 = [1.5, 2, 3, PI]
print a1+b1

[[ 2.5          4.          6.          7.14159265]]
[1, 2, 3, 4, 1.5, 2, 3, 3.141592653589793]
```

A diferencia de otros lenguajes, el operador * se aplica elemento a elemento en los arrays NumPy. El producto de matrices se realiza utilizando la función dot:

```
A = np.array( [[1, 1], [0, 1]] )
B = np.array( [[2, 3], [1, 4]] )
print A*B
print A.dot(B)
print np.dot(A,B)

[[2 3]
 [0 4]]
[[3 7]
 [1 4]]
```

```
[[3  7]
 [1  4]]
```

7.1.2. Funciones elementales

NumPy contiene las funciones matemáticas elementales con sus nombres habituales. Por ejemplo

```
print np.sin(PI/2)
print np.exp(-1)
print np.arctan(np.inf)
print np.sqrt(4)


1.0
0.367879441171
1.57079632679
2.0
```

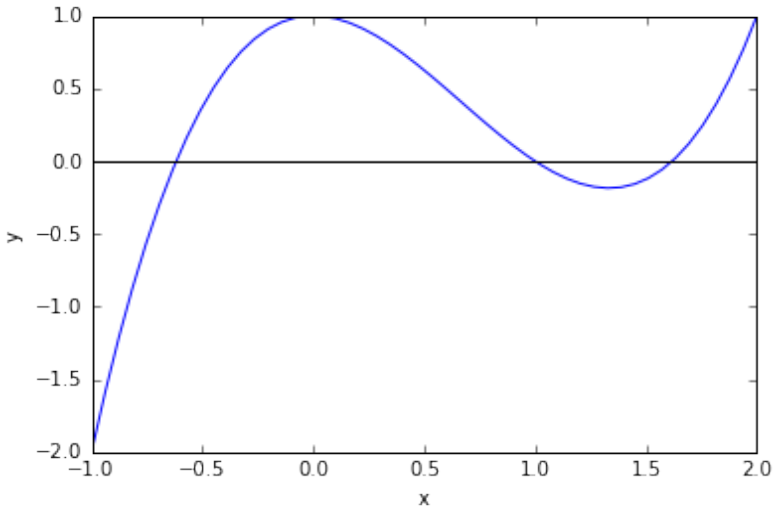
7.2. matplotlib

Usamos el paquete Matplotlib para hacer plots sencillos. La segunda línea del código siguiente se usa sólo en Notebooks pero no en línea de comando o en Spyder.

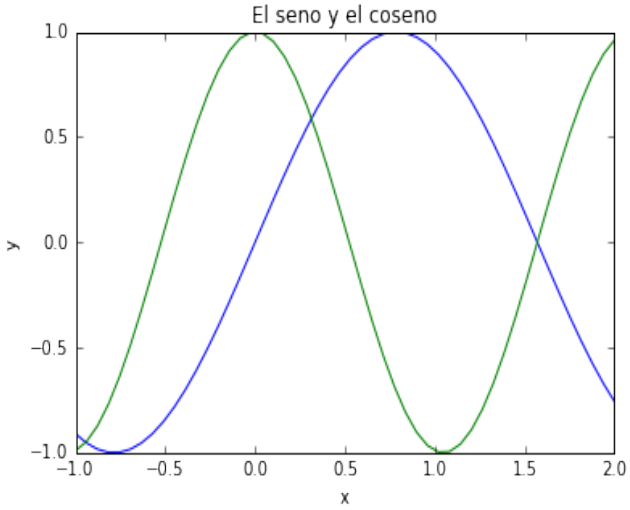
```
import matplotlib.pyplot as plt
%matplotlib inline
```

Plot en una dimensión.

```
plt.figure(1)
plt.plot(x,f(x))                    # dibujar la funcion
plt.plot(x,0*x,'k-')                # dibujar el eje X
plt.xlabel('x')
plt.ylabel('y')
plt.show()
```




```
plt.figure(2)
plt.plot(x,np.sin(2*x),x,np.cos(3*x))
plt.xlabel('x')
plt.ylabel('y')
plt.title('El seno y el coseno')
plt.show()
```

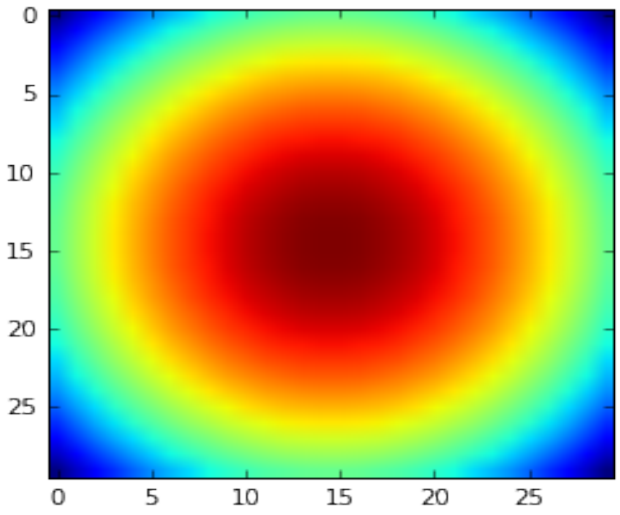


Plot en dos dimensiones

```
# Malla en el cuadrado [-1,1]x[-1,1]
x = np.linspace( -1, 1, 30)      # columnas (Ancho)
y = np.linspace( -1, 1, 30)      # filas (alto)

[X,Y] = np.meshgrid(x,y)
A = np.exp(-(X**2+Y**2)/10)

plt.imshow(A);
```



7.3. scipy