MANUAL PROGRAMACIÓN PYTHON 3.4

V1 Julio 2014

AUTOR: CARLOS VIZCARRA LUGO

INTRODUCCIÓN

Historia de Python

Python se deriva directamente del lenguaje de script ABC, que se utilizó para la enseñanza en la década de los 80s. La aparición de Python fue provocado por la necesidad de desarrollar herramientas para automatizar tareas monótonas y laboriosas.

Guido van Rossum es el creador de Python. Él comenzó a trabajar en Python a finales de 1989 en el CWI en Ámsterdam para que se empezara a utilizar en los años 90s. El nombre del lenguaje proviene de la afición de Guido por los humoristas británicos Monty Python.

Python puede actuar como un lenguaje de conexión que une a muchos de los componentes de software independientes de una manera sencilla y flexible. También se utiliza como un lenguaje de guía para operaciones de bajo nivel, de alto nivel y de control de módulos, implementados con bibliotecas de otros lenguajes de programación.

Características

Python es un lenguaje de programación que se encuentra formado por una serie de librerías y utilerías. Es un lenguaje formal con sus propias reglas específicas y formatos y es conocido por su simplicidad y flexibilidad,

Permite utilizar desde Inicio su Intérprete de línea de comandos (>>>) para acelerar el aprendizaje, ejecutar, depurar, y probar de forma interactiva lo que se programa.

Python es uno de los lenguajes de programación más fáciles para aprender y utilizar, es utilizado por muchos programadores profesionales y por las principales empresas a nivel mundial como Google, el New York Stock Exchange, Industrial Light and Magic entre otros.

El Código reutilizable es grande, y sigue la filosofía de “un módulo compartible es mejor”. Al compartir su código a toda la comunidad Python, no solo se comparte código sino también conocimiento y experiencia.

Funciona en un esquema de programación orientada a objetos (POO) en la legibilidad, coherencia y calidad del software, en general fue diseñado para ser legible y reutilizable.

Python aumenta la productividad del desarrollador que otros lenguajes de programación más complejos como C++, y Java. Python genera un tercio menos de código que los lenguajes anteriormente mencionados.

La mayoría de los programas de Python se ejecutan sin cambios en las principales plataformas informáticas, los Scripts de Python pueden portarse fácilmente,

Python viene con una gran colección de pre-compilados y funcionalidad portátil, conocido como la biblioteca estándar. Además existe una gran variedad de bibliotecas públicas que el usuario puede instalar de manera gratuita.

Python se aplica en diferentes campos como en la Programación de Juegos y Multimedia con el sistema de pygame, La comunicación de puerto serial en Windows, Linux con la extensión PySerial, Tratamiento de imágenes con PIL, PyOpenGL, Blender, Maya, Robótica con el kit de herramientas PyRo, Análisis XML con el paquete Xml y el módulo xmlrpclib, La programación de inteligencia artificial con simuladores de redes neuronales y shells de sistemas expertos, Análisis de lenguaje natural con el paquete NLTK.

Como un sistema popular de "Código Abierto" (Open Source), Python goza de una gran comunidad de desarrolladores que responden a preguntas y desarrollan mejoras a una mayor velocidad que muchos desarrolladores de software comercial. Es completamente libre de usar, es portátil y gratis.

La implementación estándar de Python está escrita en el lenguaje C, compila y funciona en prácticamente cualquier plataforma por ejemplo, los programas de Python funcionar hoy en todo sistema operativo como Unix, Microsoft Windows, DOS, MAC, BeOS, VMS, QNX entre otros. También funciona en diferentes plataformas de hardware desde PDAs y iPods hasta grandes servidores HP, Solaris e IBM.

Python tiene una sintaxis y semántica sencilla y utiliza pocas palabras reservadas. También Python ofrece interfaces para todas las principales bases de datos comerciales.

El presente manual tiene el objetivo de mostrar las funciones básicas de Python versión 3.4, en idioma español, con sencillos ejemplos utilizando el intérprete (>>>) para que el estudiante entienda el funcionamiento de cada instrucción.

El presente manual es una guía de referencia en español para los interesados en aprender a programar en un lenguaje sencillo y ampliamente utilizado a nivel mundial que es Python versión **3.4**, misma que es la última liberada al momento de la elaboración de este manual. Este manual utiliza bibliotecas, funciones y clases que funcionan en la mencionada versión y es posible que no todos los ejemplos funcionen en versiones anteriores,

Se recomienda utilizar el presente manual como una guía de programación en escuelas a nivel secundaria, preparatoria, en carreras técnicas y licenciaturas en los campos de Tecnología de Información, Computación, Informática, Telemática, Mecatrónica, Electrónica y especialidades afines.

Si deseas aprender solo un Lenguaje de Programación con una amplia gama de opciones y un gran futuro, te recomiendo **Python**.

INSTALACIÓN Y HERRAMIENTAS

La página principal de Python en español es https://wiki.python.org/moin/SpanishLanguage
Ahí encontraras toda la información que necesitas para iniciarte en este lenguaje de programación.

Para baja el paquete de Python 3.4, dirígete a la siguiente liga:
https://www.python.org/ftp/python/3.4.1/python-3.4.1.msi

Una vez que bajes el ejecutable, solo córrelo (doble clic) y se instalará Python de manera sencilla.
Estoy considerando que te encuentras trabajando sobre MS Windows.

Las librerías externas que deberás instalar para realizar los ejercicios en este manual son:
graphics.py Paquete de funciones gráficas diseñado para hacer que sea fácil dibujar gráficos.
http://mcsp.wartburg.edu/zelle/python/graphics.py

Tcl/Tk - tkinter.py Es un extenso paquete con funciones gráficas (GUI) de Python.
http://prdownloads.sourceforge.net/tcl/tcl861-src.zip

Pillow.py La librería Pillow es un conjunto de utilerías que facilitan el manejo de imágenes.
http://www.lfd.uci.edu/~gohlke/pythonlibs/

Numpy Módulo que maneja funciones matemáticas.
http://www.lfd.uci.edu/~gohlke/pythonlibs/#numpy

Pygame Es una biblioteca diseñada para que sea fácil programar aplicaciones gráficas y software multimedia.
http://pygame.org/ftp/pygame-1.9.1release.win32-py2.5.exe

También se recomienda instalar un “Editor de Programas”, puedes elegir cualquiera de los siguientes:

NotePad++	http://notepad-plus-plus.org/
Geany	http://www.geany.org/
SciTE	http://www.scintilla.org

OPERADORES ARITMÉTICOS

Operadores Básicos

Suma	**+**
Resta	**-**
División	**/**
División Entera	**//**
División Residuo	**%**
Multiplicación	*****
Cuadrado	****

Realizar los siguientes ejemplos dentro del intérprete (**>>>**) Python:

```
>>>
>>> 5 + 3
8
>>> 25 - 15
10
>>> 27 / 3
9.0
>>> 29 % 3
2
>>> 8 * 4
32
>>> 2 ** 3
8
>>> 5 + 3 * 8
29
>>> 10 - 8 + (5 * 2) + (150 / 15)
22.0
>>> (4 * 5) + (3 * 2)
26
>>> 6 + -4
2
>>> -8 + -3
-11
>>> -7 - -5
-2
```

Operadores adicionales que soporta Python son Números Complejos y Números Flotantes, no cubiertos en este manual.

CADENA DE CARACTERES

Una “Cadena de Caracteres” está formada por uno o varios caracteres que incluyen letras, números, caracteres especiales y signos.
En Python a los caracteres alfanuméricos se les coloca doble comilla, por ejemplo “Hola Buenos Días”. Los caracteres numéricos no tienen doble comilla.

Longitud de una cadena de caracteres: para obtener el valor numérico que indica el número de caracteres en una cadena de caracteres, se utiliza la función **len(x)**, ejemplos:

```
>>> len("Buenos Dias")
11
>>> len("Sabado y Domingo")
16
```

Nota: No funciona con cadena de caracteres numéricos.

Concatenar Cadena de Caracteres: es posible juntas dos o más cadenas de caracteres con el operador **+** de la siguiente forma:

```
>>> "Hot" + "Dog"
'HotDog'
>>> "Nombre " + "Apellido"
'Nombre Apellido'
>>> "Tengo " + "20" + " años"
'Tengo 20 años'
```

Repetición de Cadena de Caracteres: Python permite repetir una cadena de caracteres n veces de la siguiente manera: **“caracteres” * n**, ejemplos:

```
>>> "Rojo " * 3
'Rojo Rojo Rojo '
>>> "He " * 5
'He He He He He '
```

Cortar Caracteres: es posible truncar el contenido de una cadena de caracteres mediante la utilización de índices para obtener determinada información como a continuación se muestra:

```
>>> frutas = "Manzana Uva Durazno Zarzamora Fresa"
>>> len(frutas)
35
>>> print(frutas[2])
n
>>> print(frutas[3])
z
```

```
>>> print(frutas[0])
M
>>> print(frutas[:17])
Manzana Uva Duraz
>>> print(frutas[:18])
Manzana Uva Durazn
>>> print(frutas[:19])
Manzana Uva Durazno
>>> print(frutas[8:])
Uva Durazno Zarzamora Fresa
>>> print(frutas[8:11])
Uva
>>> print(frutas[:])
Manzana Uva Durazno Zarzamora Fresa
```

El índice de una cadena de caracteres comienza en 0.

También se puede utilizar un índice negativo para obtener una determinada cadena de caracteres:

```
>>> frutas = "Manzana Uva Durazno Zarzamora Fresa"
>>> len(frutas)
35
>>> print(frutas[:-5])
Manzana Uva Durazno Zarzamora
>>> print(frutas[8:-15])
Uva Durazno
```

Una forma de obtener la posición (índice) de una serie de caracteres (ocurrencia) en una cadena de es mediante las funciones **find()** y **index()**.

La función **find()** regresa la posición (índice) en la que se encontró la serie de caracteres buscados (considerando la bucqueda de izquierda a derecha):

```
>>> frutas = "Manzana Uva Durazno Zarzamora Fresa"
>>> print(frutas.find("Z"))
20
>>> print(frutas.find("o"))
18
>>> print(frutas.find("ana"))
4
>>> frutas.find("azno")
15
```

Si no se encuentra la ocurrencia, la función **find()** entrega como resultado **-1**.

```
>>> frutas = "Manzana Uva Durazno Zarzamora Fresa"
>>> frutas.find("K")
-1
```

La función **rfind()** despliega la posición donde se encontró la ocurrencia de derecha a izquierda.

```
>>> frutas = "Manzana Uva Durazno Zarzamora Fresa"
>>> frutas.find("Z")
20
>>> frutas.rfind("Z")
20
```

Las funciones **index()** e **rindex()** funcionan igual que **find()** y **rfind()** respectivamente, con la única diferencia que cuando no encuentran la ocurrencia a buscar, ambas funciones despliegan un mensaje de error:

```
>>> frutas = "Manzana Uva Durazno Zarzamora Fresa"
>>> frutas.index("K")
Traceback (most recent call last):
  File "<stdin>", line 1, in <module>
ValueError: substring not found
```

Funciones que modifican el contenido de una cadena de caracteres en mayúsculas y minúsculas.

upper() convierte la cadena de caracteres a mayúsculas.

capitalize() convierte la cadena de caracteres en minúsculas, menos la letra de la primera palabra.

lower() convierte la cadena de caracteres a minúsculas.

swapcase() convierte las mayúsculas en minúsculas y viceversa.

title() maneja la cadena de caracteres como un título, coloca mayúsculas al inicio de cada palabra..

Ejemplos:

```
>>> mensaje = "MARIA y JOSE son de Merida Yucatan"
>>> mensaje.upper()
'MARIA Y JOSE SON DE MERIDA YUCATAN'
>>> mensaje.capitalize()
'Maria y jose son de merida yucatan'
>>> mensaje.lower()
'maria y jose son de merida yucatan'
>>> mensaje.swapcase()
'maria Y jose SON DE mERIDA yUCATAN'
>>> mensaje.title()
'Maria Y Jose Son De Merida Yucatan'
```

La función que reemplaza el contenido de una cadena de caracteres por otra es **replace(viejo, nuevo)**.

```
>>> mensaje = "MARIA y JOSE son de Merida Yucatan"
>>> resultado = mensaje.replace("Merida", "Progreso")
>>> print(resultado)
MARIA y JOSE son de Progreso Yucatan
```

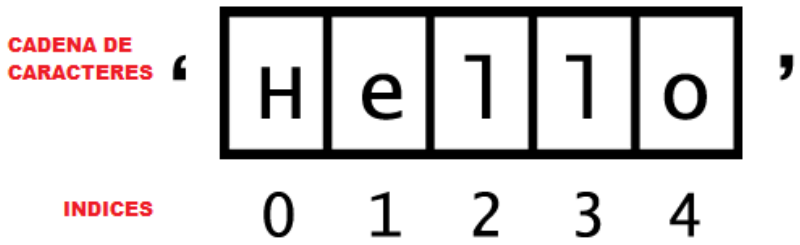
La función **expandtabs(n)** inserta un número determinado **n** de Tabs (espacios) al final de la cadena.

```
>>> expande = mensaje.expandtabs(5)
>>> print(expande)
MARIA y JOSE son de Merida Yucatan                    |
```

Index

Es frecuente que se requiera un solo carácter de una cadena de caracteres, para lograrlo Python maneja el concepto de índices o en ingles index. El índice indica la posición en donde se encuentra el carácter deseado, y para conocer la posición Python utiliza los corchetes para y dentro de ellos el número de índice que se desea extraer de la cadena de caracteres.

Es importante mencionar que la posición del índice comienza con el número 0 para el carácter de la primera posición, el número 1 para el carácter de la segunda posición y así sucesivamente. Si el índice es negativo entonces indica a partir de la última posición, es decir, el valor de índice -1 corresponde al último carácter de la cadena, el -2 corresponde al penúltimo carácter de la cadena, u así sucesivamente.



Ejemplos:

```
>>> nom = "Juan"
>>> print(nom[2])
a
>>> print(nom[0])
J
>>> print(nom[-1])
n
>>> print(nom[-3])
u
```

Para obtener más de un carácter, es decir un número determinado de caracteres dentro de una Cadena de Caracteres, se establece el rango de índices con el símbolo dos puntos **(x:y)**, sin olvidar que el índice siempre comienza con cero. El primer número indica a partir de qué posición toma los caracteres, el segundo indica a partir de qué posición corta los caracteres restantes. Ejemplos:

```
>>> alumno = " Juan Carlos Perez"
>>> len(alumno)
18
>>> print(alumno[5:12])
 Carlos
>>> print ("Hola a Todos como se encuentran"[7:13])
Todos
>>> print ("Hola a Todos como se encuentran"[-11:])
 encuentran
>>> print ("Hola a Todos como se encuentran"[-11:-5])
 encue
```

Caracteres de Escape:

Los principales caracteres de escape son:

****	Backslash (\)
\'	Comilla simple (')
\"	Comilla doble (")
\t	Tab

Los caracteres de escape se utilizan para no confundir a Python cuando se manejan caracteres reservados dentro de una cadena de caracteres (string). Ejemplos:

```
>>> print("Hola a \"Todos\" los jovenes")
Hola a "Todos" los jovenes
>>> print("Mi nombre es \t \'Juan\'")
Mi nombre es     'Juan'
```

Terminación de Líneas

Diferentes sistemas Operativos permiten manejar caracteres para terminar una línea.

En Windows para terminar una línea (Return) utiliza **\n** y **\r** para marcar el final de una línea. Otros Sistemas Operativos como Linux utilizan solo **\n**. Ejemplo:

```
>>> print("Manejo de Espacios \n \n \n Arriba de este mensaje")
Manejo de Espacios


 Arriba de este mensaje
>>>
```

VARIABLES

Al realizar programas es frecuente almacenar de manera temporal valores y caracteres alfanuméricos para manipular y obtener los resultados deseados.

Las variables son unos de los conceptos más importantes en la programación Python. También son llamadas etiquetas de variables o punteros de variables.

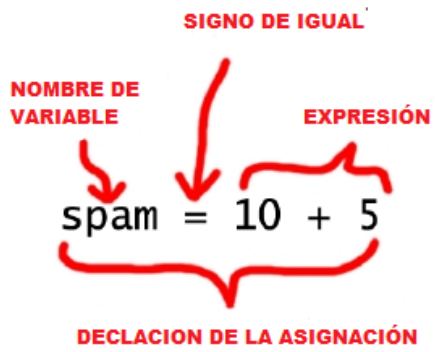
Una Variable es una cadena de caracteres al que se le asigna un determinado número de caracteres.

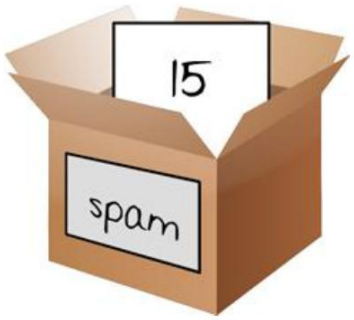
Python no distingue si los caracteres que se le asigno a una determinada variable son numéricos o alfanuméricos, nosotros debemos precisar el tipo para poder manipular adecuadamente el contenido de la variable.

Se asigna el valor a una variable con el operador **=**

Supongamos que a la variable **spam** se le ha asignado el resultado de la operación 10 + 5:

```
>>> spam = 10 + 5
>>> spam
15
```





El valor asignado a una variable puede ser modificado en cualquier momento.

```
>>> spam = 15
>>> spam + 5
20
>>> spam = 3
>>> spam + 5
8
>>> Costo_bruto = 1500
>>> print (Costo_bruto)
1500
```

Se pueden utilizar tantas variables como se deseen, teniendo cuidado en no utilizar nombres reservados que Python utiliza en su lenguaje de programación. Ejemplo de nombres reservados son: **then, so, if, sqrt, etc**.

Reglas para definir una Variable:

- No utilizar caracteres especiales.
- No utilizar el espacio.
- Es posible utilizar la línea baja "_".
- No existe un límite máximo de caracteres, pero es recomendable no utilizar nombres grandes.
- Sensitiva a las letras mayúsculas y minúsculas.
- Es recomendable asignar los nombres de variables acorde con el tipo de información o uso.

Se recomienda usar variables para asignar de manera temporal algún resultado que posteriormente será utilizado.

Asignación Múltiple

Cuando manejamos variables, generalmente utilizamos la asignación básica, es decir hace más que asignar el resultado de una sola expresión a una única variable.

Python permiten la asignación múltiple, es decir, asignación de múltiples variables a la vez. El lado izquierdo y derecho debe tener el mismo número de elementos.

Por ejemplo, la siguiente secuencia de comandos tiene varios ejemplos de asignación múltiple.

```
>>> x, y, z = 11, 25, 42
>>>
>>> print(x)
11
>>> print(y)
25
>>> print(z)
42
>>>
>>> Pedro, Maria = "Matematicas", "Biologia"
>>> print(Pedro)
Matematicas
>>> print(Maria)
Biologia
```

FUNCIONES

A continuación se mostrará la utilización de diversas funciones útiles para elaborar programas. Las funciones que se muestran a continuación se encuentran en la librería estandar que Python reconoce desde su instalación y no es requerido importarlas.

La ventaja que tiene el lenguaje de programación Python con respecto a otros es su sencillez, flexibilidad y la posibilidad de utilizar funciones predefinidas en librerías personales y públicas. Las funciones se comparten y se reutilizan a nivel personal, empresarial o en internet.

La utilización de funciones simplifica la programación, "*Si alguien más ya programó un función que requiero ejecutar, para que lo vuelvo a programar*"

Las principales funciones que tiene Python en su librería son las siguientes:

print()
La función Print se utiliza para imprimir cadenas de caracteres, números, listas o resultados.
Su sintaxis es print(X), donde X tiene una cadena de caracteres, números, una lista o el resultado de una operación.
Es una función muy utilizada dentro del intérprete Python para ir depurando y verificando lo que se programa.

```
>>> print("Mi nombre es Juan")
Mi nombre es Juan
>>> print(55 + math.pow(2,3))
63.0
>>> xx = 20
>>> print("Yo tengo ", xx, "años de edad")
Yo tengo  20 años de edad
>>> print()

>>> print("Juan" + "Pedro" + "Luis")
JuanPedroLuis
>>> print(25 + 65 - (250 - 30))
-130
>>> y = (math.log(25) + 45 ** 3) - (math.tan(32) * 10)
>>> print (y)
91121.60881541003
```

int.()

La función int() convierte cadena(s) de carácter(es) en número(s) entero(s). Es muy utilizada esta función cuando se obtiene una cadena de caracteres a través del tecleado y se quieren convertir a números para efectuar una operación matemática.

```
>>> int("15")
15
>>> print(int("45"))
45
>>> print (35 + int("45"))
80
```

round()

La función round(number[, ndigits]) redondea el número *number* al número determinado en *ndigits*.

```
>>> print(10 / 30)
0.3333333333333333
>>> print(round(10 / 30, 4))
0.3333
```

abs()

La función abs/(x) regresa el valor absoluto del número x, utilizado para asegurar que el resultado deseado sea positivo.

```
>>> abs(-1)
1
>>> v = 10 - 25
>>> print (v)
-15
>>> print (abs(v))
15
```

bin()

La función bin(x) convierte el número x en binario.

```
>>> print(bin(8))
0b1000
>>> print(bin(255))
0b11111111
```

oct()

La función oct(x) convierte el número x en Octagonal.

```
>>> print(oct(512))
0o1000
>>> print(oct(1354))
0o2512
```

hex()

La función hex(x) convierte el número x en Hexadecimal.

```
>>> print(hex(255))
0xff
>>> print(hex(125))
0x7d
```

max()

La función max(arg1, arg2, *args[, key]) regresa el argumento que tenga mayor valor definido con respecto al resto de los argumentos.

```
>>> print(max(1, 2, 9, 4, 3))
9
>>> print(max(1, 2, 9, 4, 7, 11, 3))
11
```

min()

La función min(arg1, arg2, *args[, key]) regresa el argumento que tenga menor valor definido con respecto al resto de los argumentos.

```
 >>> print(min(1, 2, 9, 4, 3))
1
>>> print(min(9, 4, 7, 11, 3, 5))
3
```

sorted()

La función sorted(iterable[, key][, reverse]) es una ponderosa función que se utiliza para ordenar de menor a mayor y viceversa, una cadena de caracteres o lista. Esta función tiene un mayor complejo funcionamiento al utilizar Listas.

```
>>> Gastos = (10.4, 8.58, 14.33, 2.35, 9.50)
>>>
>>> sorted(Gastos)
[2.35, 8.58, 9.5, 10.4, 14.33]
```

input(x)

La función input permite asignar valores a partir del teclado a una variable, si sintaxis es **x = input(y)**

Ejemplos:

```
>>> Nom = input("Cual es tu nombre: ")
Cual es tu nombre: Pedro
>>> Ape = input("Cual es tu apellido: ")
Cual es tu apellido: Lopez
>>> print("Hola ", Nom + " " + Ape + " espero que tengas un buen dia")
Hola  Pedro Lopez espero que tengas un buen dia
```

Otro ejemplo:

```
>>> num01 = input("Piensa en un numero: ")
Piensa en un numero: 12
>>> num22 = input("piensa en otro número: ")
piensa en otro número: 35
>>> print("La suma de los dos números es: ", int(num01) + int(num22))
La suma de los dos números es:  47
>>> xyz = input("Teclea un número: ")
Teclea un número: 25
>>> resultado = int(xyz)
>>> print("El Logaritmo del numero ",xyz, " es ", math.log(resultado))
El Logaritmo del numero  25  es  3.2188758248682006
```

Mayúsculas

Para convertir una cadena de caracteres a mayúsculas se utiliza la función **upper()** con la sintaxis *mensaje. upper()*

```
>>> nom = “juan”
>>> print (nom.upper())
JUAN
>>> print ("hola a todos".upper())
HOLA A TODOS
```

Minúsculas

Para convertir una cadena de caracteres a minúsculas se utiliza la función **lower()** con la sintaxis *mensaje. lower()*

```
>>> z = "MARIA"
>>> z = z.lower()
>>> print(z)
Maria
>>> print ("COMO TE LLAMAS?".lower())
como te llamas?
```

find()

La función **find(x)** regresa la posición en donde se encuentra x en la cadena de caracteres. Si no se encuentra el carácter, regresa como valor -1.

```
>>> Palabra = "Parangaricutirimicuaro"
>>> len(Palabra)
22
>>> Palabra.find("g")
5
>>> Palabra.find("y")
-1
```

Strip()

La función strip permite remover espacios dentro de una cadena de caracteres que se encuentran en la parte derecha de la cadena, como se muestra a continuación.

```
>>> print ("Como estan todos?      ".strip())
Como estan todos?
```

Statistics

Permite utilizar diversas funciones estadísticas como:

Media	statistics.mean()
Mediana	statistics.median()
Valor más bajo de la Mediana	statistics.low()
Valor más alto de la Mediana	statistics.high()
Moda	statistics.mode()

```
>>> import statistics
>>> datos = [10, 15, 12, 21, 11, 17, 18, 14, 25, 16]

>>> print("La Media es: ", statistics.mean(datos))
La Media es:  15.9

>>> print("La Mediana es: ", statistics.median(datos))
La Mediana es:  15.5

>>> print("La Moda es: ", statistics.mode([1, 9, 2, 8, 5, 7, 4, 6, 5]))
La Moda es:  5
```

Para conocer todas las funciones que Python 3.4 instala de manera estandar, ir a la liga;
https://docs.python.org/3/library/index.html

Funciones Locales

Python también permite definir funciones a nivel programa, mismos que pueden ser utilizados en diversas ocasiones sin afectar el flujo del programa. Una función local se caracteriza porque realiza una función específica pero el programa requiere ejecutarlo varias veces. Las funciones Locales permiten la recursividad, es decir ejecutar simultáneamente en un programa varias funciones,

La sintaxis de una función local en Python es la siguiente:

def *nombre* (parametro, resultado):
    instrucciones
    return resultado

Los siguientes ejemplos deben ser tecleados en el editor de programas, para posteriormente ejecutarlo.

```
def imprime_lista(lista):
# imprime el contenido de la lista
    print("Los componentes de la lista son:")
    for index in range(len(lista)):
        print("Elemento ", index)


def Suma (lista)
# suma el contenido de la lista y regresa el resultado
total = 0
for j in range(len(lista)):
    total = total + j
return total


def Numero_Factorial(numero):
    resultado = numero
    for indice in range(1,n):
        resulto *= indice
    return resultado
```

LIBRERÍAS

Las librerías contienen una serie de funciones que alguien más ha realizado para que al importarlas, podamos utilizarlas. Para poder usar estas librerías, es requerido instalarlas una sola vez para que Python pueda reconocerlas y es requerido cargarlas a memoria cada vez que se utilicen.

math
Librería **math** tiene varias funciones Matemáticas
Una vez instalada la librería, **math,** es requerido cargarla a memoria de la siguiente manera:

```
>>> import math
```

Una vez cargada la librería en memoria se pueden utilizar las siguientes funciones matemáticas:

sqrt(x)	raíz cuadrada de x
pow(x,y)	x a la potencia de y
factorial(x)	factorial del número x
log(x,y)	Logaritmo de x en base b
log(x)	Logaritmo de x en base e
exp(x)	e a la potencia x
degree(x)	Convierte x de radianes a grados
radians(x)	convierte x de grados a radianes
ceil(x)	Seno del número x
cos(x)	Coseno del número x
tan(x)	Tangente del número x

En Python, para utilizar cualquier función de una determinada librería, la sintaxis que se utiliza es **librería.funcion**.

Ejemplos:

```
>>> math.sqrt(4)
2.0
>>> math.sqrt(4) * math.sqrt(10 * 10)
20.0
>>> math.log(25 * 5)
4.8283137373023015
>>> math.factorial(5)
120
>>> math.pow(2, 4)
16.0
```

graphics.py

La librería **graphics.py** es un paquete con funciones diseñada para fácitar dibujar gráficos.

```
>>>
>>> import graphics
>>>
>>> graphics.GraphWin("Circulo", 100, 100)   (Abre una ventana con un circulo)
<graphics.GraphWin object at 0x02B78550>
>>>
```

Para mostrar el contenido de la librería graphics.py, se utiliza el comando **dir()** como a continuación se muestra:

```
>>> dir(graphics)
['BAD_OPTION', 'Circle', 'DEAD_THREAD', 'DEFAULT_CONFIG', 'Entry', 'GraphWin', 'GraphicsError',
'GraphicsObject', 'Image', 'Line', 'OBJ_ALREADY_DRAWN', 'Oval', 'Point', 'Polygon', 'Rectangle', 'Text', 'Transform',
'UNSUPPORTED_METHOD', '_BBox', '__builtins__', ' __cached__', '__doc__', '__file__', '__loader__', '__name__',
'__package__', '__spec__', '__version__', '_root', 'color_rgb', 'os',  'sys', 'test', 'time', 'tk', 'update']
```

Solo es requerido invocar la librería (import graphics) una sola vez en el programa o en la sesión de la terminal.

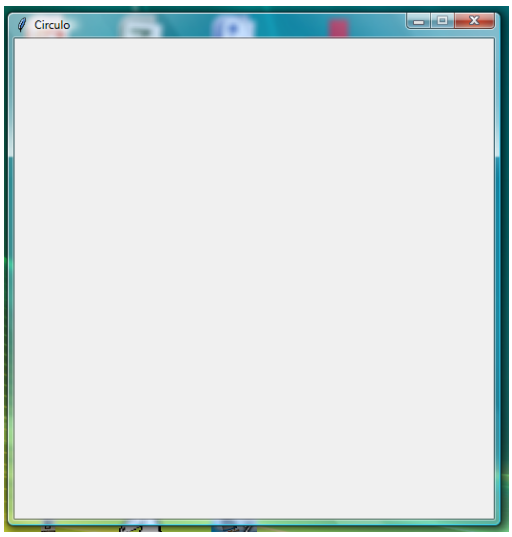
En la línea de commandos, teclear las siguientes instrucciones y revisar el resultado final:

```
>>> x = graphics.GraphWin("Circulo", 500, 500)
>>> y = graphics.Circle(graphics.Point(150,150), 10)
>>> y.draw(x)
>>> x.close()
```

Describiendo paso a paso el resultado del bloque de instrucciones anterior, el desplegado gráfico es el siguiente:

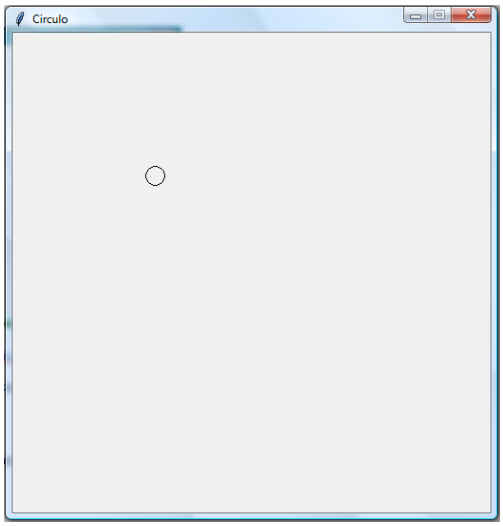
```
>>> x = graphics.GraphWin("Circulo", 500, 500)
```

Genera en la pantalla de la computadora la siguiente ventana:



```
>>> y = graphics.Circle(graphics.Point(150,150), 10)
>>> y.draw(x)
```

Las anteriores instrucciones definen la figura del círculo y lo dibuja dentro de la pantalla.




```
>>>x.close()
```

Esta instrucción cierra la ventana.

GraphWin

Esta función **GraphWin(Titulo, ancho, alto)** permite crear una ventana sobre la pantalla de la computadora. Los parámetros que se pueden utilizar son muy completos y tiene una gran variedad de opciones para crear y configurar pantallas.

Para ver todos los objetos y parámetros que tiene GraphWin, teclear en la línea de comandos lo siguiente:

```
>>> dir(graphics.GraphWin())
```

Se recomienda realizar los siguientes ejercicios para comprender mejor el funcionamiento de dicha función gráfica.

Realiza una ventana:

```
>>> ventana = graphics.GraphWin("MI VENTANA", 800, 500)
```

Genera una ventana y en ella coloca un **punto** usando la función **Point(x,y)**:

```
>>> import graphics
>>> ventana = graphics.GraphWin()
>>> punto = graphics.Point(100, 50)
>>> punto.draw(ventana)
>>> ventana.close()
```

Genera una ventana y en ella dibuja una **línea** con la función **Line(punto1, punto2)**, con lados opuestos en los punto1 y punto2:

```
>>> import graphics
>>> ventana = graphics.GraphWin()
>>> linea = graphics.Line(20, 10)
>>> linea.draw(ventana)
>>> ventana.close()
```

Genera una ventana, coloca un punto y dibuja un **círculo** alrededor del punto:

```
>>> import graphics
>>> ventana = graphics.GraphWin()
>>> punto = graphics.Point(100, 50)
>>> punto.draw(ventana)
>>> circulo = graphics.Circle(punto, 25)
>>> circulo.draw(ventana)
>>> ventana.close()
```

Ahora se le coloca color al círculo, su perímetro es rojo y su interior es azul:

```
>>> import graphics
>>> ventana = graphics.GraphWin()
>>> punto = graphics.Point(100, 50)
>>> punto.draw(ventana)
>>> circulo = graphics.Circle(punto, 25)
>>> circulo.setOutline('red')
>>> circulo.setFill('blue')
>>> circulo.draw(ventana)
>>> ventana.close()
```

Genera una ventana y dibuja un **rectángulo** con la función **Rectangle(punto1, punto2)**, con los vértices de los lados opuestos en los punto1 y punto2, con perímetro negro e interior verde:

```
>>> import graphics
>>> ventana = graphics.GraphWin("DIBUJO DE UN RECTANGULO", 800, 500)
>>> rectangulo = graphics.Rectangle(graphics.Point(20, 10), graphics.Point(100, 50))
>>> rectangulo.setOutline('black')
>>> rectangulo. setFill('yellow')
>>> rectangulo.draw(ventana)
>>> ventana.close()
```

Genera una ventana y mediante el mouse se seleccionan 3 puntos para dibujar un triángulo mediante la función de polígono que es **Polygon(punto1, punto2, punto3, ...)**

```
>>> import graphics
>>> “”” Crea una Ventana con fondo amarillo “””
>>> ventana = graphics.GraphWin("DIBUJA UN RECTANGULO", 800, 500)
>>> ventana.setBackground('yellow')
>>> mitexto = "Haz click en 3 puntos"
>>> centro = int(ventana.getWidth() / 2)
>>> mensaje = graphics.Text(graphics.Point(centro, 60), mitexto)
>>> mensaje.setTextColor('red')
>>> mensaje.setStyle('italic')
>>> mensaje.setSize(20)
>>> mensaje.draw(ventana)
>>> “”” Con el Mouse establece los 3 vertices del triangulo “””
>>>  punto1 = ventana.getMouse()
>>> punto1.draw(ventana)
>>> punto2 = ventana.getMouse()
>>> punto2.draw(ventana)
>>> punto3 = ventana.getMouse()
>>> punto3.draw(ventana)
>>> vertices = [punto1, punto2, punto3]
>>> “”” Dibuja el triangulo “””
>>> triangulo = graphics.Polygon(vertices)
>>> triangulo.setFill('gray')
>>> triangulo.setOutline('cyan')
>>> triangulo.setWidth(4)
>>> triangulo.draw(ventana)
>>> “”” Para terminar el programa al hacer click en la ventana “””
>>> mesaje.setText(“Haz click para salir”)
>>> ventana.getMouse()
>>> ventana.close()
```

Otro ejemplo

```
>>> import graphics
>>> ventana = graphics.GraphWin('Una Cara', 200, 150) # Crea una ventana
>>> encadezado = graphics.Circle(graphics.Point(40,100), 25) # Se define un encabezado
>>> centrar = graphics.Circle(graphics.Point(40,100), 25) # Define posición en la ventana
>>> centrar.setFill("yellow") # Relleno del circulo en amarillo
>>> centrar.draw(ventana) # Dibuja el circulo
```

```
>>> ojo1 = graphics.Circle(graphics.Point(30, 105), 5) # Define la posición del ojo
>>> ojo1.setFill("blue") # Color azul
>>> ojo1.draw(ventana) # Dibuja el ojo
>>> ojo2 = graphics.Line(graphics.Point(45, 105), graphics.Point(55, 105)) # Define posicion del otro ojo
>>> ojo2.setWidth(3)
>>> ojo2.draw(ventana) # Dibuja el otro ojo
>>> boca = graphics.Oval(graphics.Point(30, 90), graphics.Point(50, 85)) @Define posicion de la boca
>>> boca.setFill("red")   # Se define color rojo
>>> boca.draw(ventana) # Se dibuja la boca
>>> mensaje = graphics.Text(graphics.Point(100, 120), "CARITA") # Define un mensaje de Texto
>>> mensaje.draw(ventana) # Dibuja el texto
```

El programa anterior genera el siguiente resultado:

Para finalizar el programa se agregan las siguientes instrucciones: Define un mensaje para salir mediante latilización del mouse y finalmente cierra la ventana:

```
>>> anuncio = graphics.Text(graphics.Point(ventana.getWidth()/2, 20), "Haz Click para Salir")
>>> anuncio.draw(ventana)
>>> ventana.getMouse()
<graphics.Point object at 0x0280BE90>
>>> ventana.close()
```

urllib

Para accesar información (texto) desde Internet, se utiliza la función, **urllib.**

```
>>> import urllib
>>> import urllib.request
>>> url = "http://www.gutenberg.org/files/2554/2554-8.txt"
>>> solicita = urllib.request.Request(url)
>>> responde = urllib.request.urlopen(solicita)
>>> pagina = responde.read()
>>> print(pagina[:90])
b'The Project Gutenberg EBook of Crime and Punishment, by Fyodor Dostoevsky\r\n\r\nThis eBook is'
```

Las anteriores instrucciones lo que hace es leer el archivo de texto ubicando en internet y llamado **2554-8.txt**, almacenando de manera temporal la información en **pagina**, para que finalmente despliegue en pantalla las últimas **90** letras de su contenido.

tkinter

tkinter es un extenso paquete estandar con funciones gráficas (GUI) de Python.

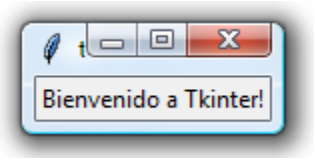
Para utilizar **tkinter** cargarlo de la siguiente manera: `>>> import tkinter` ó `>>> from tkinter import *`

A continuación se muestran una serie de ejemplos que ayudan a comprender la utilización de las librerias que se utilizan en **tkinter**:

Crea una ventana con mensaje de bienvenida:

```
>>> from tkinter import *                                  # Importa las librerias de Tkinter
>>> ventana = Tk()                                         # Inicializa Tkinter y crea una ventana
>>> etiqueta = Label(ventana, text="Bienvenido a Tkinter!") # Se define una etiqueta a la ventana
>>> etiqueta.pack()                                        # Crea a la ventana con texto
>>> ventana.mainloop()                                     # La ventana se mantendra hasta cerrarla
```

El resultado de las siguientes instrucciones es:



A continuación se muestra otro ejemplo pero generando dos botones en la ventana:

```
>>> from tkinter import *                                  # Importa las librerias de Tkinter
>>> marco = Frame()                                        # Inicializa Tkinter y crea una ventana
>>> marco.pack()                                           # Asocia la ventana
>>> boton = Button(marco, text="Salir", fg="red", command=marco.quit) # Define boton con texto rojo
```

```
>>> boton.pack(side=LEFT)          # Coloca el boton anterior a la izquierda
>>> texto = Button(marco, text="Hola")   # Define el boton con texto Hola
>>> texto.pack(side=LEFT)          # Coloca el boton anterior a la izquierda
>>> marco.mainloop()               # Genera un ciclo infinto hasta que
                                   # Oprimar el boton "Salir
```

Genera la siguiente ventana:

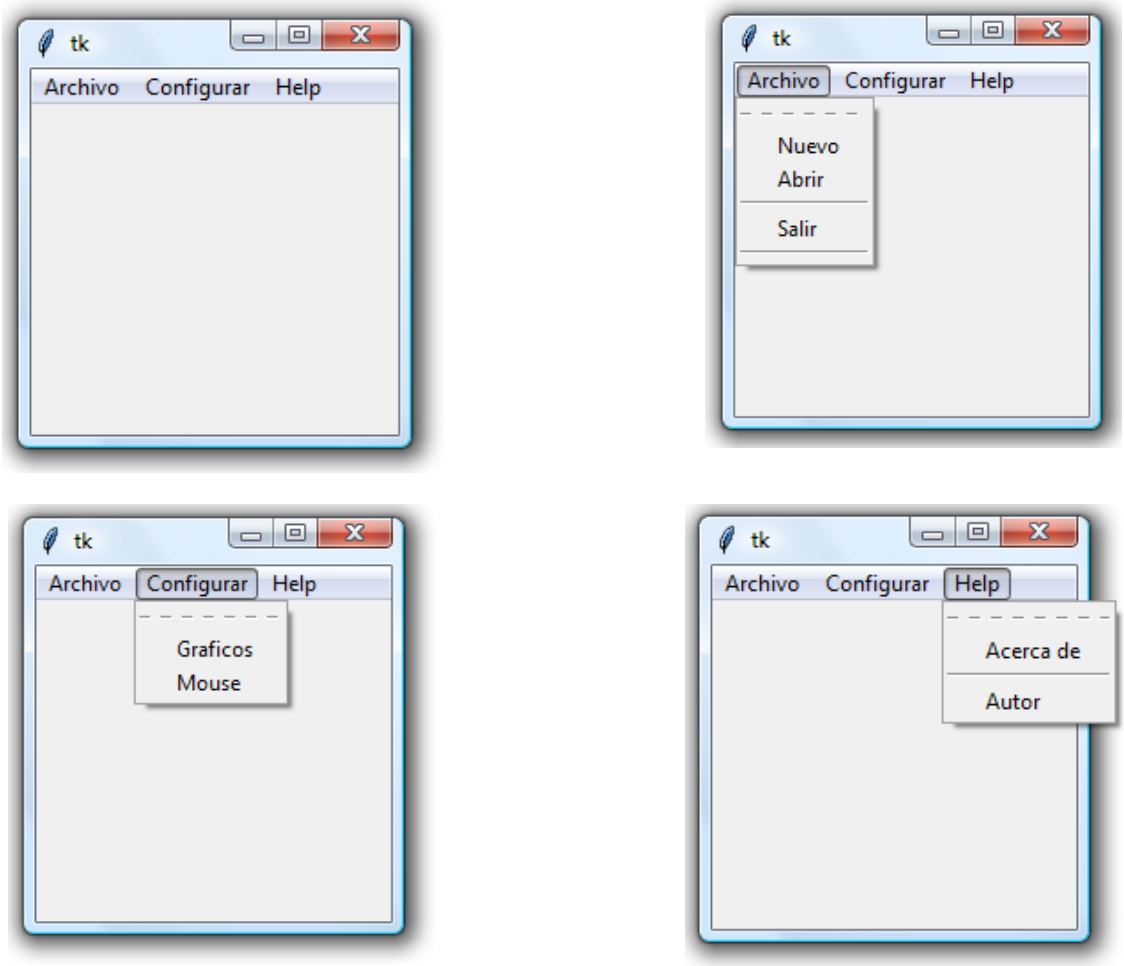
Puedes oprimir el boton "**Hola**" tantas veces como lo desee y el programa no terminará, Cuando oprimes el boton "**Salir**", fijate que regresa el pronpt "**>>>**" del interprete a la pantalla. Revisar paso a paso cómo se va modificando la ventana generada para comprender el funcionamiento de las isntrucciones.

Crea una Ventana con varios Menús:

Para este ejemplo, ir tecleando cada instrucción en el intérprete y revisar el resultado que va generando en la ventana, de esa manera entenderán cómo funciona cada instrucción.

```
>>> from tkinter import *
>>> principal = Tk()
>>> menu = Menu(principal)
>>> principal.config(menu=menu)
>>> menu_archivo = Menu(menu)
>>> menu.add_cascade(label="Archivo", menu=menu_archivo)
>>> menu_archivo.add_command(label="Nuevo")
>>> menu_archivo.add_command(label="Abrir")
>>> menu_archivo.add_separator()
>>> menu_archivo.add_command(label="Salir")
>>> menu_trabajo = Menu(menu)
>>> menu.add_cascade(label="Configurar", menu=menu_trabajo)
>>> menu_trabajo.add_command(label="Graficos")
>>> menu_trabajo.add_command(label="Mouse")
>>> menu_ayuda = Menu(menu)
>>> menu.add_cascade(label="Help", menu=menu_ayuda)
>>> menu_ayuda.add_command(label="Acerca de")
>>> menu_ayuda.add_separator()
>>> menu_ayuda.add_command(label="Autor")
>>> mainloop()
```

El Desplegado que genera es el siguiente:



Es posible asociarle una función a cada instrucción **.add_command()** para que ejecute una acción determinada mediante el comando **command=** como a continuación se muestra:

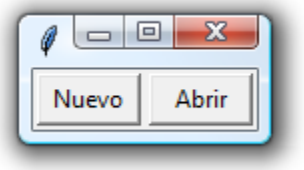
menu_archivo.add_command(label="Nuevo", command=letrero)
Al elegir de opción **Nuevo** de la ventana **Archivo**, se ejecutará el proceso llamado **letrero.**

Teclear el siguiente programa en un editor, salvarlo y ejecutarlo para revisar su contenido. También se recomienda utilizar la función Debugger para revisar instrucción por instrucción.

```
from tkinter import *
def letrero():
        print("Ejecuta Opcion")
def salir():
        sal = input("Pulsa Enter para Salir!")
```

```
principal = Tk()
menu = Menu(principal)
principal.config(menu=menu)
menu_archivo = Menu(menu)
menu.add_cascade(label="Archivo", menu=menu_archivo)
menu_archivo.add_command(label="Nuevo", command=letrero)
menu_archivo.add_command(label="Abrir", command=letrero)
menu_archivo.add_separator()
menu_archivo.add_command(label="Salir", command=salir)
menu_trabajo = Menu(menu)
menu.add_cascade(label="Configurar", menu=menu_trabajo)
menu_trabajo.add_command(label="Graficos", command=letrero)
menu_trabajo.add_command(label="Mouse", command=letrero)
menu_ayuda = Menu(menu)
menu.add_cascade(label="Ayuda", menu=menu_ayuda)
menu_ayuda.add_command(label="Acerca de, command=letrero")
menu_ayuda.add_separator()
menu_ayuda.add_command(label="Autor", command=letrero)
mainloop()
```

A continuación se muestra como crear una Ventana de Herramientas como la siguiente:




```
from tkinter import *
root = Tk()
toolbar = Frame(root)
b = Button(toolbar, text="Nuevo", width=6)
b.pack(side=LEFT, padx=2, pady=2)
b = Button(toolbar, text="Abrir", width=6)
b.pack(side=LEFT, padx=2, pady=2)
toolbar.pack(side=TOP, fill=X)
mainloop()
```

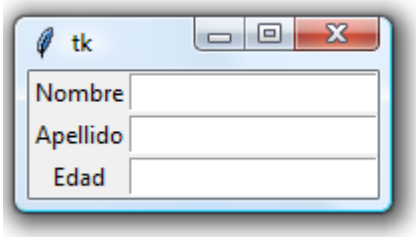
Por último, la mayoría de las aplicaciones utilizan una barra de estado en la parte inferior de cada ventana de la aplicación. La implementación de una barra de estado con **tkinter** es trivial: sólo tiene que utilizar un la instrucción **Label** como a continuación se muestra:

```
class StatusBar(Frame):
   def __init__(self, master):
      Frame.__init__(self, master)
      self.label = Label(self, bd=1, relief=SUNKEN, anchor=W)
      self.label.pack(fill=X)
   def set(self, format, *args):
      self.label.config(text=format % args)
      self.label.update_idletasks()
   def clear(self):
      self.label.config(text="")
      self.label.update_idletasks()
status = StatusBar(root)
status.pack(side=BOTTOM, fill=X)
```

Las siguientes instrucciones crean una Ventana de Diálogo, para la captura de información:

```
>>> from tkinter import *
>>> ventana = Tk()
>>>
>>> Label(ventana, text="Nombre").grid(row=0)
>>> Label(ventana, text="Apellido").grid(row=1)
>>> Label(ventana, text="Edad").grid(row=2)
>>>
>>> e1 = Entry(ventana)
>>> e2 = Entry(ventana)
>>> e3 = Entry(ventana)
>>>
>>> e1.grid(row=0, column=1)
>>> e2.grid(row=1, column=1)
>>> e3.grid(row=2, column=1)
>>> mainloop()
```

La ventana generada es:



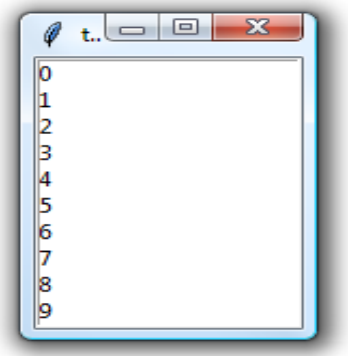
Como pudieron observar, la instrucción **.grid()** genera el área de captura dentro de la pantalla.

Ahora se muestra como desplegar información dentro de una pantalla:

```
>>> from tkinter import *
>>> ventana = Tk()
>>> lista = Listbox(ventana)
>>> lista.pack()
>>> for i in range(20): lista.insert(END, str(i))
>>> mainloop()
```

Se puede observar que dentro de la pantalla generada es posible imprimir caracteres de información según se requiera, en este ejemplo se imprime de manera consecutiva los número del 0 al 10 usando la función **for.**

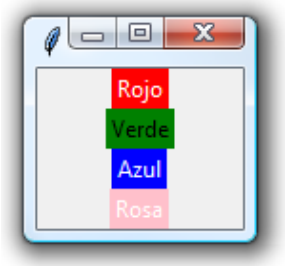
El resultado es:



Teclea las siguientes instrucciones y sigue paso a paso el resultado generado:

```
>>> from tkinter import *
>>> ventana = Tk()
>>> colores = Label(ventana, text="Rojo", bg="red", fg="white")
>>> colores.pack()
>>> colores = Label(ventana, text="Verde", bg="green", fg="black")
>>> colores.pack()
>>> colores = Label(ventana, text="Azul", bg="blue", fg="white")
>>> colores.pack()
>>> colores = Label(ventana, text="Rosa", bg="pink", fg="white")
>>> colores.pack()
>>> mainloop()
```

El resultado es:



Ahora teclea las siguientes instrucciones y compara el resultado con el ejemplo anterior:

```
>>> from tkinter import *
>>> pantalla = Tk()
>>> x = Label(pantalla, text="Rojo", bg="red", fg="white")
>>> x.pack(fill=X)
>>> x = Label(pantalla, text="verde", bg="green", fg="black")
>>> x.pack(fill=X)
>>> x = Label(pantalla, text="Azul", bg="blue", fg="white")
>>> x.pack(fill=X)
>>> x = Label(pantalla, text="Rosa", bg="pink", fg="white")
>>> x.pack(fill=X)
>>> mainloop()
```

El cambio es que se fijaron los colores en toda la columna.

También es posible despliegar la información por columna:

```
>>> from tkinter import *
>>> ventana = Tk()
>>> color = Label(ventana, text="Rojo", bg="red", fg="white")
>>> color.pack(side=LEFT)
>>> color = Label(ventana, text="Verde", bg="green", fg="black")
>>> color.pack(side=LEFT)
>>> color = Label(ventana, text="Azul", bg="blue", fg="white")
>>> color.pack(side=LEFT)
>>> color = Label(ventana, text="Rosa", bg="pink", fg="white")
>>> color.pack(side=LEFT)
>>> mainloop()
```

Genera el resultado:

Pillow

La librería PIL.py tiene un conjunto de utilerías que facilitan el manejo de imágenes en Python.

Para importar la librería **Pillow**, se teclea:

```
>>> from PIL import Image
```

No olvidar que la librería PIL.py debió ser instalada previamente.

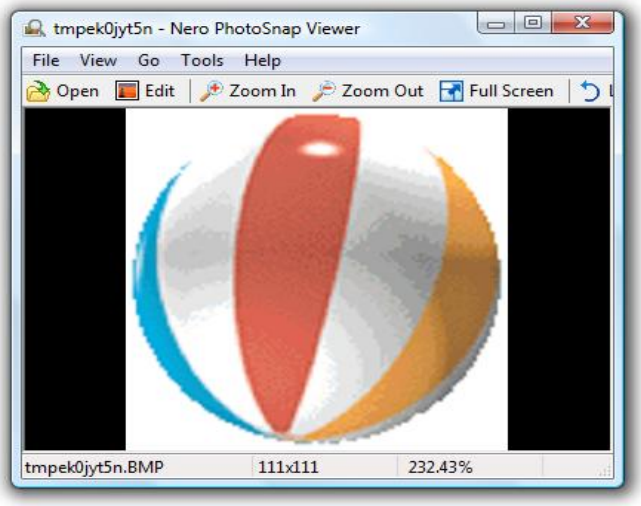
Para cargar una imagen;

```
>>> imagen = Image.open("ball.gif")
```

En donde el archivo **ball.gif** se encuentra ubicado en el mismo subdirectorio en donde se instaló **Python**.

Para revisar el contenido de la imágen:

```
>>> from __future__ import print_function
>>> print(imagen.format, imagen.size, imagen.mode)
GIF (111, 111) P
>>> imagen.show()
```



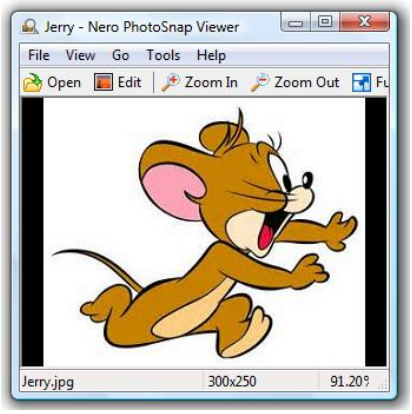
Para convertir una imagen a “JPEG thumbnails” (pequeña representación gráfica), editar y probar el siguiente programa:

```
from __future__ import print_function
import os, sys
from PIL import Image
size = (128, 128)
for indice in sys.argv[1:]:
    outfile = os.path.splitext(indice)[0] + ".thumbnail"
    if indice != outfile:
        try:
            im = Image.open(indice)
            im.thumbnail(size)
            im.save(outfile, "JPEG")
        except IOError:
            print("No se puede crear el Thumbnai para", indice)
```

Para rotar una imagen 45 grados:

```
>>> from PIL import Image
>>> imagen = Image.open("Jerry.jpg")
>>> imagen.rotate(45).show()
```

El resultado es: Antes Despues

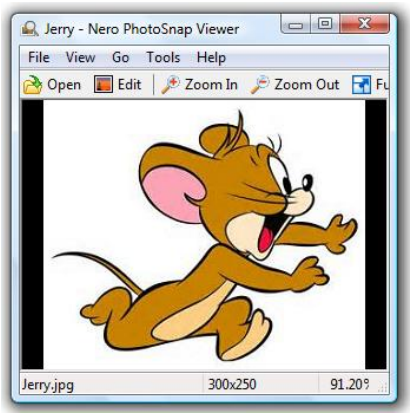


El módulo que te permite escribir sobre **imágenes 2D** es ImageDraw, ejemplo:

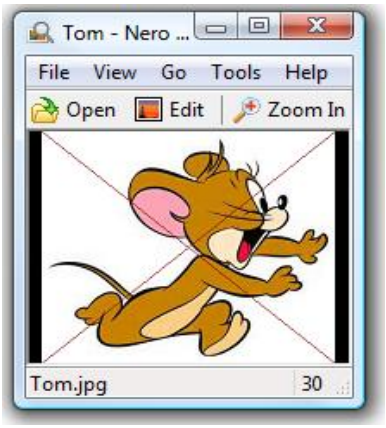
```
>>> from PIL import Image
>>> from PIL import ImageDraw
>>> imagen = Image.open("Jerry.jpg")
>>> dibuja = ImageDraw.Draw(imagen)
>>> dibuja.line((0, 0) + imagen.size, fill=128)
>>> dibuja.line((0, imagen.size[1], imagen.size[0], 0), fill=128)
>>> del dibuja
>>> imagen.save("Tom.jpg", "PNG")
```

Como resultado de las instrucciones anteriores, se genera el siguiente archivo:

Antes: Jerry.jpg



Despues: Tom.jpg



El archivo original lo modificó dibujando unas líneas cruzadas y generando un nuevo archivo.

La librerías gráficas anteriores tienen una gran número de funciones para la manipulación de imágenes, por lo que si se encuentra interesado en conocerlas al detalle, accese la siguiente página:
http://pillow.readthedocs.org/en/latest/guides.html

Numpy

Librería que contiene funciones matemáticas y graficas. Permite trabajar con arreglos, matrices y gráficos de una manera más sencilla que las funciones estándares de Python.

Numpy es utilizada en conjunto con otras funciones para el correcto funcionamiento de algunas de sus funciones. Por los que se recomienda que se instalen también las siguientes librerías: SciPy y Matplotlib, Pyparsing, Dateutil, Six y Pytz.

La función **array** se utiliza para crear un arreglo como se muestra:

```
>>> import numpy as np
>>> x = np.array([2, 3, 1, 0])
>>> print (x)
[2 3 1 0]
```

Para crear un arreglo de 3 x 3 llamado índices teclear lo siguiente:

```
>>> np.indices((3,3))
array([[[0, 0, 0],
        [1, 1, 1],
        [2, 2, 2]],
```

```
       [[0, 1, 2],
        [0, 1, 2],
        [0, 1, 2]]])
```

Crear un arreglo de 2 x 3 llamado indices:

```
>>> np.indices((2,3))
array([[[0, 0, 0],
        [1, 1, 1]],

       [[0, 1, 2],
        [0, 1, 2]]])
```

Ahora crear con un editor de texto el archivo que se llame arreglo.txt, colocar la siguiente información y salvarlo en el mismo directorio en donde se instaló Python 3.4, al teclear el contenido del archivo se verá así:

```
55 11 99
22 66 77
88 33 44
```

En el intérprete de Python y teclear lo siguiente:
Omitir incluir en el intérprete los comentarios (#), solo las instrucciones.

```
>>> import numpy                          # Cargar librería numpy a memoria
>>> import io                             # Cargar librería io a memoria

>>> archivo=open("arreglo.txt", "r")      # Abre y lee arreglo.txt y se carga a la variable “archivo”
>>> datos=archivo.read()                  # Lee y asigna el contenido de “archivo” en “datos”
>>> archivo.close()                       # Cierra el archivo para evitar que se dañe
>>> print(datos)                          # Imprime la informacion asociada a “datos”
55 11 99
22 66 77
88 33 44

>>> arre = numpy.genfromtxt(io.BytesIO(datos.encode()))    # Se crea un arreglo a partir de “datos”
>>> print(arre)
[[ 55.  11.  99.]
 [ 22.  66.  77.]
 [ 88.  33.  44.]]
```

Para obtener determinada información del arreglo **arre**, se utiliza el concepto de **Índice**, como se muestra:

```
>>> print(arre[0])
[ 55.  11.  99.]
>>> print(arre[1])
[ 22.  66.  77.]
>>> print(arre[2])
[ 88.  33.  44.]
>>>
```

```
>>> print(arre[1,1])
66.0
>>> print(arre[2,0])
88.0
>>> print(arreglo[0,2])
99.0
```

A continuación se convierte el arreglo en una matriz mediante la función **matrix()**:

```
>>> lamatriz = numpy.matrix(arre)
>>>
```

La variable **lamatriz** es una matriz con el contenido del arreglo llamado "**arre**".
Es importante convertir nuestros arreglos en matrices para poder manipular la información, es decir, realizar una operación aritmética como suma o multiplicación de matrices por ejemplo.

Si se imprime el contenido de **lamatriz** en la pantalla, el desplegado es parecido al arreglo **arre**, sin embargo para Python es una matriz que se encuentra lista para que sea manipulada.

```
>>> print(lamatriz)
[[ 55.  11.  99.]
 [ 22.  66.  77.]
 [ 88.  33.  44.]]
```

Ahora, que pasa si a la matriz **lamatriz**, se le multiplica por 2 y su resultado se le asigna a **multiplica**:

```
>>> multiplica = lamatriz * 2
>>> print(multiplica)
[[ 110.   22.  198.]
 [  44.  132.  154.]
 [ 176.   66.   88.]]
```

multiplica contiene el resultado de la operación anterior.

Ahora se muestra cómo convertir un arreglo a partir de una cadena de caracteres, para ello, teclear lo siguiente:

```
>>> info = "31 10 52\n11 36 12\n22 67 43"
```

Nota el carácter de escape **\n** que permite delimitar la terminación de la fila en el arreglo.

Convertir **info** a un arreglo:

```
>>> res = numpy.genfromtxt(io.BytesIO(info.encode()))
>>>
>>> res
array([[ 31.,  10.,  52.],
       [ 11.,  36.,  12.],
       [ 22.,  67.,  43.]])
```

Finalmente convertir el arreglo **res** a matriz:

```
>>> aa = numpy.matrix(res)
>>>
>>> aa
matrix([[ 31.,  10.,  52.],
        [ 11.,  36.,  12.],
        [ 22.,  67.,  43.]])
```

Otra forma de crear una matriz, es mediante la función **matrix** como se muestra:

```
>>> A = numpy.mat("1 2 3; 4 5 6; 7 8 9")
>>>
>>> A
matrix([[1, 2, 3],
        [4, 5, 6],
        [7, 8, 9]])
```

Para realizar una transposición de la matriz **A**, usar el atributo **T**:

```
>>> Trans = A.T
>>> print(Trans)
[[1 4 7]
 [2 5 8]
 [3 6 9]]
```

Para invertir la matriz **A**, usar el atributo **I**:

```
>>> invert = A.I
>>> print(invert)
[[ -4.50359963e+15   9.00719925e+15  -4.50359963e+15]
 [  9.00719925e+15  -1.80143985e+16   9.00719925e+15]
```

En vez de crear una matriz a partir de una cadena de caracteres, se puede crear a partir de un arreglo:

```
>>> nuevo = numpy.mat(numpy.arange(9).reshape(3, 3))
>>>
>>> print(nuevo)
[[0 1 2]
 [3 4 5]
 [6 7 8]]
```

Para la realizar la división entre dos matrices, se utiliza el atributo **div**:

```
>>> a = numpy.array([2, 6, 5])
>>> b = numpy.array([1, 2, 3])
>>> div = numpy.divide(b, a)
>>> print(div)
[ 0.5        0.33333333  0.6       ]
```

También la librería **numpy** tiene Funciones Financieras:
la función **fv** calcula el valor futuro, dando el valor del instrumento financiero y el valor futuro.
La función **pv** calcula el valor presente.
La función **npv** regresa el valor presente neto.
La función **rate** calcula la tasa de interés.

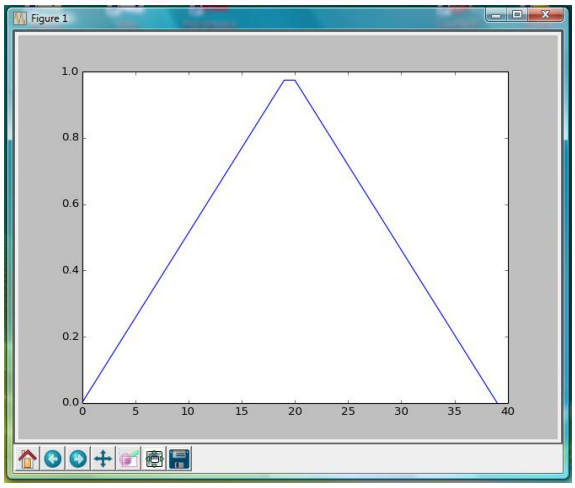
Como ejemplo consideremos que se realizará un préstamo de$1,000 (valor actual), con un interés del 3%, pagos trimestrales de 10 por 5 años. Se desea calcular el valor futuro:

```
>>> ValorFuturo = numpy.fv(0.03/4, 5 * 4, -10, -1000)
>>> print("El Valor Futuro es: ", ValorFuturo)
El Valor Futuro es:  1376.09633204
```

Las funciones de ventana (Windows) que maneja **numpy**, son utilizadas para desplegar funciones matemáticas para analizar el comportamiento de datos, las principales son: Bartlett, Hamming, Káiser y Sinc.

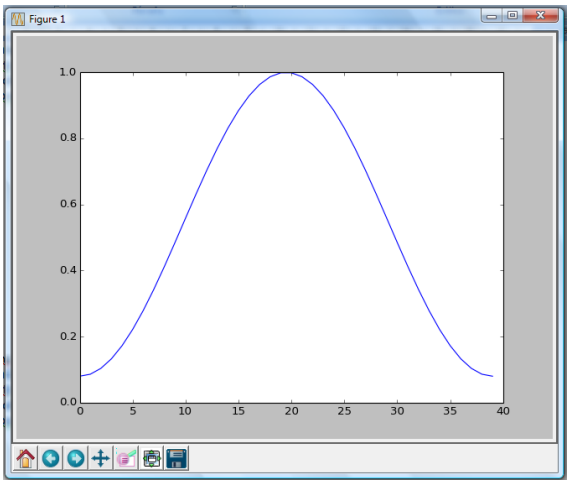
Bartlett genera una ventana triangular:

```
>>> import numpy
>>> ventana = numpy.bartlett(40)
>>> plot(ventana)
[<matplotlib.lines.Line2D object at 0x04199EF0>]
>>> show()
```



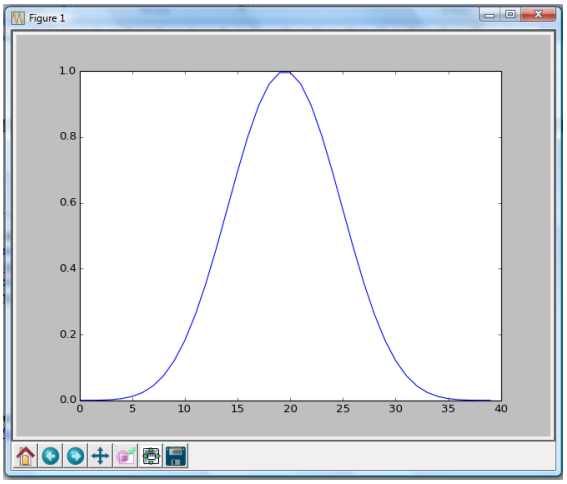
Hamming calcula una gráfica de campana:

```
>>> vent2 = numpy.hamming(40)
>>> plot(vent2)
[<matplotlib.lines.Line2D object at 0x042CC4D0>]
>>> show()
```



Káiser despliega un desplegado de función tipo Kaiser:

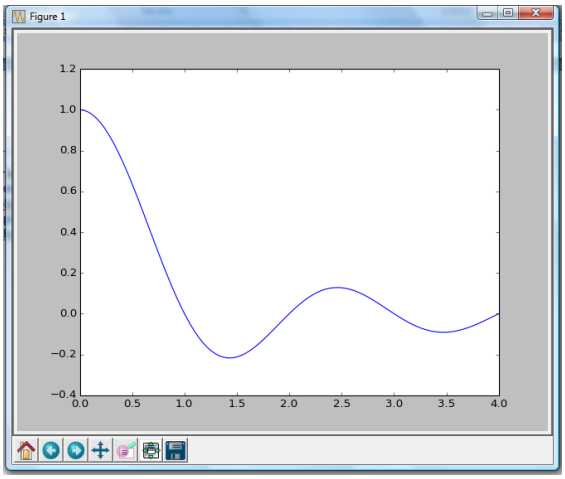
```
>>> vent3 = numpy.kaiser(40, 14)
>>> plot(vent3)
[<matplotlib.lines.Line2D object at 0x040573F0>]
>>> show()
```



Para mayor información de las ventanas de funciones mencionadas, ve la página siguiente:
http://es.wikipedia.org/wiki/Ventana_%28funci%C3%B3n%29

La función **Sinc** es muy utilizada para el procesamiento de señales / información.

```
>>> x = numpy.linspace(0, 4, 100)
>>> valores = numpy.sinc(x)
>>> plot(x, valores)
[<matplotlib.lines.Line2D object at 0x042ADC30>]
>>> show()
```



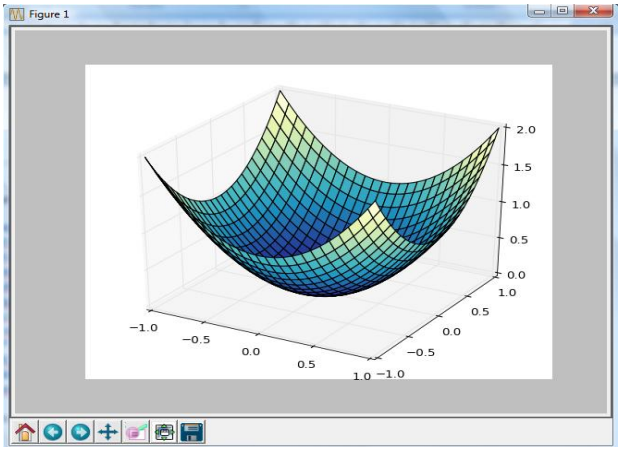
Para realizar gráficas en tercera dimensión en base a la función:

$$z = x^2 + y^2$$

Se utiliza el parámetro **projection="3d"**, como se muestra:

```
>>> from mpl_toolkits.mplot3d import Axes3D
>>> import matplotlib.pyplot as plt
>>> import numpy as np
>>> from matplotlib import cm
>>> fig = plt.figure()
>>> ax = fig.add_subplot(111, projection="3d")
>>> u = np.linspace(-1, 1, 100)
>>> x, y = np.meshgrid(u, u)
>>> z = x ** 2 + y ** 2
>>> ax.plot_surface(x, y, z, rstride=4, cstride=4, cmap=cm.YlGnBu_r)
<mpl_toolkits.mplot3d.art3d.Poly3DCollection object at 0x041A3B70>
>>> plt.show()
```

El resultado es:



Para información de funciones numpy y matplotlib, ve a la liga siguiente: http://matplotlib.org/index.html

Datetime

La función datetime permite manipular la fecha del sistema de varias maneras.

```
>>> import datetime
>>> from datetime import *
>>> FechaHoy = date.today()
>>> print(FechaHoy)
2014-05-22
```

Para formatear la fecha, teclear las siguientes instrucciones:

```
>>> FormatoFecha = date.today().strftime("%A %B %d %Y")
>>> print ("Hoy es: ", FormatoFecha)
Hoy es:  Friday May 30 2014

>>> Fecha = datetime.today().strftime("%Y-%m-%d")
>>> print("Hoy es ",Fecha)
Hoy es:  2014-05-22
>>> Hora = datetime.today().strftime("%H:%M:%S")
>>> print("Son las: ", Hora, " horas")
Son las:  20:10:36  horas
```

Otro ejemplo:

```
>>> import datetime
>>> from datetime import *
>>> edad = int(input("Cuantos años tienes? "))
Cuantos años tienes? 49
>>> anio = int(datetime.today().strftime("%Y"))          # Obtiene el año del dia de hoy del sistema
>>> nacimiento = anio - edad
>>> print("Naciste en el año ",nacimiento)
Naciste en el año  1965
```

Para desplegar el día de la semana en español, se puede utilizar el siguiente Diccionario:

```
>>> ListaDia = {"Monday": "Lunes", "Tuesday": "Martes", "Wednesday": "Miercoles", "Thursday": "Jueves",
"Friday": "Viernes", "Saturday": "Sabado", "Sunday": "Domingo"}
>>> dia = date.today().strftime("%A")   # Se obtiene el dia de la semana en ingles
>>> hoy = ListaDia[dia]                 # Se obtiene el dia en español apartir del Diccionario
>>> print("Este dia es ",hoy)
Este dia es  Viernes
```

Pygame
Es una biblioteca diseñada para que sea fácil de programar aplicaciones gráficas y software multimedia, como los juegos.

Para comenzar a utilizar la biblioteca **pygame**, inicialmente se carga a memoria:

```
>>> import pygame
```

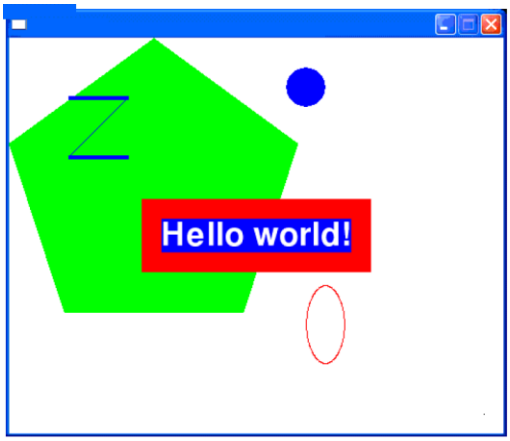
A continuación se inicializa la función:

```
>>> pygame.init()
```

Ya se está listo para realizar el siguiente ejemplo:

```
>>> import pygame, sys
>>> from pygame.locals import *
>>> pygame.init()
>>>
>>> # Define Ventana
>>> superficie = pygame.display.set_mode()
>>> pygame.display.set_caption()
>>>
>>> # Define los colores
>>> NEGRO = (0, 0, 0)
>>> BLANCO = (255, 255, 255)
>>> ROJO = (255, 0, 0)
>>> VERDE = (0, 255, 0)
>>> AZUL = (0, 0, 255)
>>>
>>> # Define Caracteristicas de las Letras
>>> basicFont = pygame.font.SysFont(None, 48)
>>>
>>> # Define el Texto
>>> text = basicFont.render('Hellow world!', True, BLANCO, NEGRO)
>>> textRect = text.get_rect()
>>> textRect.centerx = superficie.get_rect().centerx
>>> textRect.centery = superficie.get_rect().centery
>>>
>>> # Establece fondo blanco en la ventana
>>> superficie.fill(BLANCO)
>>>
>>> # Dibuja Poligono Verde
>>> pygame.draw.polygon(superficie, VERDE, ((146, 0), (291, 106), (236, 277), (56, 277), (0, 106)))
>>>
>>> # Dibuja lineas
>>> pygame.draw.line(windowSurface, AZUL, (60, 60), (120, 60), 4)
>>> pygame.draw.line(superficie, AZUL, (120, 60), (60, 120))
```

```
>>> pygame.draw.line(superficie, AZUL, (60, 120), (120, 120), 4)
>>>
>>> # Dibuja Circulo Azul
pygame.draw.circle(superficie, AZUL, (300, 50), 20, 0)
>>>
>>> # Dibuja un Elipse Rojo
>>> pygame.draw.ellipse(superficie, ROJO, (300, 250, 40, 80), 1)
>>>
>>> # Dibuja Texto con un rectangulo de fondo
>>> pygame.draw.rect(superficie, ROJO, (textRect.left - 20, textRect.top - 20, textRect.width + 40, textRect.height + 40))
>>>
>>> # Dibuja un Arreglo
>>> pixArray = pygame.PixelArray(superficie)
>>> pixArray[480][380] = NEGRO
>>> del pixArray
>>>
>>> # Dibuja Texto
>>> windowSurface.blit(text, textRect)
>>>
>>> # Dibuja la ventana en la pantalla
>>> pygame.display.update()
```



Teclea las siguientes instrucciones en un editor y correlo:

```
import pygame, sys
from pygame.locals import *
pygame.init()
# Define el Movimiento de la Imagen
IPS = 30 # 30 imagenes por segundo
IPSReloj = pygame.time.Clock()
# Define las Caracteristicas de la ventana
ventana = pygame.display.set_mode((400, 300), 0, 32)
pygame.display.set_caption("Me Parecio ver un lindo Gatito")
```

```
BLANCO = (255, 255, 255)
gato = pygame.image.load("C:\Python34\gato.png") # Carga la imagen del archivo gato.png
gatotx = 10
gatoty = 10
direction = "derecha"
while True:                          # Ciclo infinito Principal
    ventana.fill(BLANCO)
    if direction == "derecha":
        gatotx += 5
        if gatotx == 280:
            direction = "down"
    elif direction == "down":
        gatoty += 5
        if gatoty == 220:
            direction = "left"
    elif direction == "left":
        gatotx -= 5
        if gatotx == 10:
            direction = "up"
    elif direction == "up":
        gatoty -= 5
        if gatoty == 10:
            direction = "derecha"
    ventana.blit(gato, (gatotx, gatoty))
    for indice in pygame.event.get():
        if indice.type == QUIT:
            pygame.quit()
            sys.exit()
    pygame.display.update()
    IPSReloj.tick(IPS)
```

·El resultado es el siguiente: (En movimiento infiinito)

CONTROL DE FLUJO

Se llama control de flujo al orden en el que se ejecutan las instrucciones de un programa, siendo las propias instrucciones las que determinan o controla dicho flujo. En un programa, a menos que el flujo de control se vea modificado por una instrucción de control, las instrucciones siempre se ejecutan secuencialmente, una detrás de otra, en orden de aparición, de izquierda a derecha y de arriba abajo, que es el flujo natural de un programa.

En Python para realizar el control de ejecución de un programa se utilizan las siguientes instrucciones:

Condicional If / elif/ else

Las declaraciones **if / elif /else** son una estructura de control, que nos permiten tomar una decisión al interior del programa, determinando que acciones se tomarán en base al "cumplimiento / no cumplimiento" de una determinada condición.

Las signos de condición que se utilizan son **==, !=, <, > , <=, >=.**

La sintaxis de la condicional en Python es la siguiente:

```
if condicion_1:
        instrucciones_1
elif condicion_2:
        instrucciones_2
elif condicion_3:
        instrucciones_3
else:
        instrucciones_4
```

Ejemplo:

```
>>> edad = input("Teclea tu edad: ")
Teclea tu edad: 20
>>> if edad < 2: print("Eres un Bebe")
... elif   3 <  edad < 18: print("Eres un Adolescente")
... elif 19 < edad < 35: print("Ya eres Mayor")
... else: print("Ya estas Viejo")
...
Ya eres Mayor
```

Ciclos - Loops

Los ciclos o Loops son utilizados para repetir un determinado número de veces las instrucciones que lo contienen. Python utiliza dos tipos de ciclos (loops) **for** y **while**.

Ciclo for:

Las instrucciones se ejecutarán *índice* número de veces.

Sintaxis:

for *índice* **in** *condición*:

    Instrucciones

Ejemplo:

```
>>> numeros = [0, 1, 2, 3, 4, 5, 6, 7, 8, 9]
>>> print(numeros)
[0, 1, 2, 3, 4, 5, 6, 7, 8, 9]
>>> for index in numeros: print(index)
0
1
2
3
4
5
6
7
8
9
```

Otro sencillo ejemplo utilizando **for** es:

La función **range(a, b)**, permite definir un rango de número entre **a** y **b**.

```
>>> for index in range(1, 10): print(index, "Mensaje")
1 Mensaje
2 Mensaje
3 Mensaje
4 Mensaje
5 Mensaje
6 Mensaje
7 Mensaje
8 Mensaje
9 Mensaje
```

Otro ejemplo más complejo es el siguiente:

```
>>> Nombres = ["Juan", "Maria", "Patty", "Jose"]
>>> Calificaciones = [6, 9, 8, 10]
>>> for index in range(len(Nombres)): print("La Calficacion de ", Nombres[index], "es ", Calificaciones[index])
La Calficacion de  Juan es  6
La Calficacion de  Maria es  9
La Calficacion de  Patty es  8
La Calficacion de  Jose es  10
```

También se utiliza la función **for** para desplegar cadenas de caracteres como se muestra:

```
>>> colores = "Rojo Verde Azul"
>>> for indice in colores: print(indice)
R
o
j
o

V
e
r
d
e

A
z
u
l
```

Ciclo while:

Sintaxis:

while *condición*:

    Instrucciones

```
numeros = [0, 1, 2, 3, 4, 5, 6, 7, 8, 9]
indice = 0
while indice < 9:
    print(numeros[indice])
    indice = indice + 1
```

ESTRUCTURAS DE DATOS

Las estructuras de datos son una colección de elementos de datos, que se encuentran ordenados (estructurados) de una determinada manera.

Cada elemento de la estructura de datos tiene asignado un número o secuencia que corresponde a su posición o índice dentro de la estructura, la primera posición o índice es el cero, la segunda posición o índice es el uno, y así sucesivamente. Esta posición o índice sirve para extraer un determinado elemento o dato de la estructura.

Ejemplo:

```
calificaciones = {
        "Juan" :  [7.5, 9.0, 8.8, 9.5, 9,1],
        "Pedro" :  [10.0, 9.8, 9.1, 9.5, 9.1],
        "Maria" :  [8.9, 9.2, 9.8, 9.5, 8.8],
        "Patty" :  [8.6, 9.5, 9.3, 8.7, 8.8 ],
}
```

Para los ejercicios siguientes se utiliza el archivo ubicado **sample.txt** ubicado en el archivo donde se instaló Python y cuyo contenido es:

Este es un ejemplo para la utilizacion de archivos.
Este es un archivo de texto.
La letra de las letras es estandar.

Si no tienes el archivo, lo puedes crear con el programa de Windows "Bloc de Notas" (Notes) y lo salvas en donde tienes instalado Python con el nombre anteriormente mencionado.

Lectura de un Archivo

La lectura de un archivo en Python se realiza con la función **open(nombre[,modo[,buffer]])**

Los argumentos que permite la función **open()** son:

r Modo lectura
w Modo escritura
a Modo para agregar
b Modo binario
+ Modo escritura y lectura

Ejemplos:

```
>>> archivo = open(r"C:\Python34\sample.txt")
>>> BD = open(+"C:\Python34\sample.txt")
>>> Awe = open("C:\Python34\sample.txt", "w")
>>> BD = open("C:\Python34\sample.txt", "a")
```

Escritura de un Archivo

El comando **write(contenido)** permite escribir información al archivo especificado.

Ejemplo:

```
>>> BD = open("C:\Python34\sample.txt", "a")
>>> Mensaje01 = “Mensaje agregado al final del archivo”
>>> Mensaje02 = “No olvides cerrar el archivo al final”
>>> BD.write(“Mensaje01)
>>> BD.write(“Mensaje02)
>>> BD.close()
```

La función **close()** cierra el archivo respectivo. Para evitar que el archivo se corrompa es necesario al final cerrar el archivo mediante la función **close()**

También se utiliza la función **read()** para leer el contenido del archivo, las siguientes funciones agregan un mensaje al inicio del archivo simple.txt. El modo “r*” permite abrir y escribir el archivo.

```
>>> f = open("C:\Python34\sample.txt", "r+")
>>> archivo = f.read()
>>> archivo = "Titulo: Mensaje Inicial" + "\n\n" + archivo
>>> f.seek(0)
0
>>> f.write(archivo)
124
>>> f.close()
>>>
```

El resultado es que modifica el archivo agregándole el mensaje cono se muestra

Titulo Mensaje Inicial
Este es un ejemplo para la utilizacion de archivos.
Este es un archivo de texto.
La letra de las letras es estandar.

La instrucción **seek(0)** mueve el puntero en el archivo a la primera posición.

Para imprimir el contenido de un archivo:

```
>>> f = open("C:\Python34\sample.txt", "r")
>>> archivo = f.read()
>>> print (archivo, end = "\n")
>>> archivo.__add__("Mensaje agregado con add")
'Titulo\n\nEste es un ejemplo para la utilizacion de archivos.\nEste es un archivo de texto.\nLa letra de las letras es estandar.
Mensaje agregado con add'
>>> katpy = archivo.__add__("\n  Mensaje agregado con add \n  \n ")
>>> print(katpy)
```

Titulo

Este es un ejemplo para la utilizacion de archivos.
Este es un archivo de texto.
La letra de las letras es estandar.
Mensaje agregado con add

Diccionarios

Es una eficiente estructura de datos para almacenar un par de valores de datos: ***llave: contenido*** dentro de ***{ }***

Ejemplo:

```
>>> Calificaciones = {"Juan": 8, "Maria": 9, "Jesus": 7, "Luis": 8, "Patty": 8}
>>> print(Calificaciones["Juan"])
8
>>>
>>> print(Calificaciones["Maria"])
9
>>> for index in Calificaciones: print(Calificaciones[index])
7
8
9
8
8
```

Existen dos restricciones en el manejo de Diccionarios:

En una estructura de Diccionario no puede tener dos contenidos iguales en la llave.
La llave de un diccionario no puede estar definida en otra lista de diccionario.

Las principales funciones en la estructura de un Diccionario son las siguientes:
d.items() muestra el contenido
d.keys() muestra la llave
d.values() muestra el valor asociado
d.get(key) regresa el valor asociado que se encuentra en **key**.
d.pop(key) remueve/extrae la llave y regresa su valor correspondiente.
d.popitem() muestra la llave y valor asociado **(key, value)** en "**d**". El índice o apuntador se va moviendo.
d.clear() remueve todos los elementos
d.copy() copia el contenido
d.fromkeys(s, t) crea un nuevo diccionario con las llaves tomadas de "**s**" y valores tomados de "**t**".
d.setdefault(key, v) si la llave "**key**" se encuentra en "**d**"
d.update(e) agrega un par (llave, valor) en e

Ejemplos:

```
>>> colores = {"azul": 45, "verde": 23, "rojo": 49, "rosa":84}
>>> for i in colores: print(colores[i])
45
23
49
84
>>> colores.pop("rojo")
49
>>> for i in colores: print(colores[i])
45
23
84
>>> print(colores)
{'azul': 45, 'verde': 23, 'rosa': 84}
>>> colores.items()
dict_items([('azul', 45), ('verde', 23), ('rosa', 84)])
>>> colores.keys()
dict_keys(['azul', 'verde', 'rosa'])
>>> colores.values()
dict_values([45, 23, 84])
>>> colores.get("verde")
23
>>> colores.popitem()
('azul', 45)
```

```
>>> colores.popitem()
('verde', 23)
>>> colores.popitem()
('rosa', 84)
>>> colores.clear()
```

Listas

Las Litas son una secuencia de elementos que se encuentran separados por comas. La utilización de las Listas es extensa ya que permiten de una manera eficiente la manipulación de grandes volúmenes de información a una alta velocidad de procesamiento. Python cuenta con una extensa lista de librerías que simplifican la manipulación de los datos y el tiempo de desarrollo de los programas.

Una Lista se construye de la siguiente manera:

Utilizando un par de corchetes **[ ]**
Mediante corchetes, separando los elementos por comas: [a], [a, b, c]
Usando una lista con la definición: [x for x in iterable]
Usando un construcción de tipo: list() o list(iterable)

Ejemplos:

```
>>> list("martes")
['m', 'a', 'r', 't', 'e', 's']
>>>
>>> dias = ["lunes", "Martes", "Miercoles", "Jueves", "Viernes"]
>>> print (dias)
['lunes', 'Martes', 'Miercoles', 'Jueves', 'Viernes']
>>>
>>> numeros = [241, 144, 560, 705, 333]
>>> print (numeros)
[241, 144, 560, 705, 333]
```

Manipulación de Listas

Las funciones que permiten manipular el contenido de las listas son las siguientes:

Funciones sort() & sorted()

Las funciónes sort() & sorted() permiten ordenar de diferentes maneras, el contenido de una lista de una manera rápida y eficiente. Ambas funciones tienen una amplia gama de opciones de acuerdo a las necesidades en manipulación del contenido de las listas. A continuación se muestran con ejemplos las utilizaciones más frecuentes:

```
>>> numeros = [241, 144, 560, 705, 333, 555, 144, 611, 999, 310]
>>> resultado = sorted(numeros)
>>> print(resultado)
[144, 144, 241, 310, 333, 555, 560, 611, 705, 999]
>>>
>>> calificaciones = [8.5, 6, 9.99, 7.45, 5.9]
>>> print(calificaciones)
[8.5, 6, 9.99, 7.45, 5.9]
>>> calificaciones.sort()
>>> print(calificaciones)
[5.9, 6, 7.45, 8.5, 9.99]
>>>
>>> dias = ["lunes", "Martes", "Miercoles", "Jueves", "Viernes"]
>>> dias.sort(key=str.lower)
>>> print(dias)
['Jueves', 'lunes', 'Martes', 'Miercoles', 'Viernes']
```

El parámetro **reverse** dentro de la función **sorted()** se utiliza para definir que el ordenamiento del contenido de la lista se realizara de manera inversa.

```
>>> caracteres = ['aa', 'BB', 'zz', 'CC']
>>> print(caracteres)
['aa', 'BB', 'zz', 'CC']
>>> print(sorted(caracteres, reverse=True))
['zz', 'aa', 'CC', 'BB']
>>> suerte = [88, 52, 10, 99, 44, 32, 15]
>>> print (suerte)
[88, 52, 10, 99, 44, 32, 15]
>>> print(sorted(suerte, reverse=True))
[99, 88, 52, 44, 32, 15, 10]
>>>
```

El parámetro **key** dentro de la función **sort(*, key=None, reverse=None)** se utiliza para definir el criterio para ordenar la Lista seleccionada como a continuación se ejemplifica.

```
>>> frutas = ["Melon", "Uva", "Fresa", "Coco", "Zalzamora", "Zandia"]
>>> print(frutas)
['Melon', 'Uva', 'Fresa', 'Coco', 'Zalzamora', 'Zandia']
>>> print(sorted(frutas, key=len))
['Uva', 'Coco', 'Melon', 'Fresa', 'Zandia', 'Zalzamora']
>>>
>>> letras = ['aa', 'BB', 'CC', 'zz']
>>> print(letras)
['aa', 'BB', 'CC', 'zz']
>>> print(sorted(letras, key=str.lower))
['aa', 'BB', 'CC', 'zz']
```

La función **zip()** es utilizada para agregar y combina los elementos de las tablas seleccionadas para formar una nueva.

La sintaxis de la función es **zip(*[interacciones(s)]*n)**

```
>>> Nombres = ["Juan", "Pedro", "Maria", "Pablo"]
>>> Calificaciones  = [7.5, 8.2, 9.5, 6.7]
>>> Resultados = zip(Nombres, Calificaciones)
 [('Juan', 7.5), ('Pedro', 8.2), ('Maria', 9.5), ('Pablo', 6.7)]
>>>
>>> Nueva = list(zip(Nombres, Calificaciones))
>>> print(Nueva)
[('Juan', 7.5), ('Pedro', 8.2), ('Maria', 9.5), ('Pablo', 6.7)]
>>>
```

Para imprimir un determinado elemento dentro de una lista se utilizan los corchetes y el número de índice que se quiere selecciona dentro de la lista.

```
>>> print(Nombres)
['Juan', 'Pedro', 'Maria', 'Pablo']
>>> print(Nombres[1])
Pedro
>>> print(Nombres[2])
Maria
>>> print(Nombres[0])
Juan
>>>
>>> print(Nueva)
[('Juan', 7.5), ('Pedro', 8.2), ('Maria', 9.5), ('Pablo', 6.7)]
>>>
>>> print(Nueva[2])
('Maria', 9.5)
```

Una forma sencilla de forma una lista a partir de otras dos es de la siguiente manera:

```
>>> nombres = ["Maria", "Pedro", "Juan"]
>>> calificacion =[9, 7, 8]
>>> resultados = [nombres, calificaciones]
>>> print(resultados)
[['Maria', 'Pedro', 'Juan'], [6, 9, 7, 5, 10]]
```

Se logra el mismo resultado si se utiliza el signo de **+**

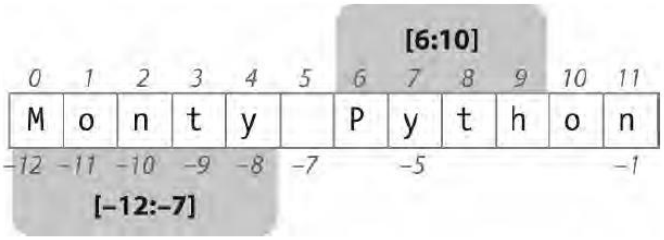
```
>>> nombres = ["Maria", "Pedro", "Juan"]
>>> calificacion =[9, 7, 8]
>>> resultados = nombres + calificaciones
>>> print(resultados)
[['Maria', 'Pedro', 'Juan'], [6, 9, 7, 5, 10]]
```

Subcadenas (substrings)
Utilizado para obtener determinados elementos dentro de una lista.

```
>>> dias = ["Lunes", "Martes", "Miercoles", "Jueves", "Viernes"]
>>> print(dias[2])
Miercoles
>>> print(dias[1:3])
['Martes', 'Miercoles']
>>> print(dias[:3])
['Lunes', 'Martes', 'Miercoles']
>>> print(dias[2:])
['Miercoles', 'Jueves', 'Viernes']
>>> print(dias[-3:])
['Miercoles', 'Jueves', 'Viernes']
>>> print(dias[-1:])
['Viernes']
>>> print(dias[:-2])
['Lunes', 'Martes', 'Miercoles']
>>> print(dias[:])
['Lunes', 'Martes', 'Miercoles', 'Jueves', 'Viernes']
```

Otro ejemplo de subcadenas: Supongamos que se tiene **nombre = "Monty Python"**

```
>>> nombre ="Monty Python"
>>> print(nombre[-12:-7])
Monty
>>> print(nombre[6:10])
Pyth
```



Las siguientes funciones se utilizan ampliamente para facilitar el manejo de las listas en Python:
append(x) agrega el elemento x al final de la lista.
count(x) regresa el número de veces que x apareción en la lista.
extend(list) agrega cada elemento de la lista en la preexistente.
index(x) regresa el valor del índice de la occurrencia x.
pop(i) remueve y regresa el elemento ubicado en el índice i.
remove(x) remueve el elemento de la ocurrencia i.
reverse() invierte el orden de los elementos.
sort() ordena en orden creciente la lista especificada.

Ejemplos:

```
>>> dias = ["Lunes", "Martes", "Miercoles", "Jueves", "Viernes"]
>>> dias.append("Sabado")
>>> print(dias)
['Lunes', 'Martes', 'Miercoles', 'Jueves', 'Viernes', 'Sabado']
>>> dias.count("M")
0
>>> dias.count("Viernes")
1
>>> dias.pop(2)
'Miercoles'
>>> print(dias)
['Lunes', 'Martes', 'Jueves', 'Viernes', 'Sabado']
>>> dias.reverse()
>>> print(dias)
['Sabado', 'Viernes', 'Jueves', 'Martes', 'Lunes']
>>> dias.sort()
>>> print(dias)
['Jueves', 'Lunes', 'Martes', 'Sabado', 'Viernes']
```

El siguiente ejemplo muestra la captura de información, la agrega a una lista, procesa la información de la lista y la guardarla en un archivo.

```
lista = [ ]
numero = 1
promedio = 0
bandera = “s”
while bandera == “s”:
        califica = int(input(“Teclea tu calificación: “))
        promedio = promedio + califica
        lista[numero - 1]  = (numero, califica)
        bandera = input(“Otra calificación? (s / n): “
        print(“Tus Calificaciones son: “, lista)
print(“Tu promedio es “ promedio / numero)
```

Asignación Múltiple

Python permite utilizar el siguiente truco para asignar diferentes valores en una sola línea:

```
>>> a, b, c = ["Rojo", "Manzana", 42]
>>> print(a)
Rojo
>>> print(b)
Manzana
>>> print(c)
42
```

EJEMPLOS

Se considera importante agregar esta sección para que el estudiante refuerce su aprendizaje de una manera más entretenida.

No porque sean juegos, se considere descartar esta sección del programa de formación académico profesional, ya que aquí el estudiante tendrá la oportunidad de manejar de una manera más compleja todo lo aprendido anteriormente.

A continuación se muestran el código de dos juegos, utilizando la biblioteca estandar de Python y módulos externos.

```
# Juego Adivina el número
import random
adivina = 0
print("Hola! Cual es tu nombre? ")
nombre = input()
numero = random.randint(1, 20)                    # Genera un numero aleatorio entre 1 y 20
print("Bueno", nombre, "Estoy pensando un numero entre 1 y 20")
print("tienes 6 intentos")
while adivina < 6:
        print("Dime un numero ")
        dato = input()
        dato = int(dato)
        adivina = adivina + 1
        if dato < numero:
                print("Tu numero es muy pequeño")
        if dato > numero:
                print("Tu numero es muy grande")
        if dato == numero:
                break
if dato == numero:
        adivina = str(adivina)
        print("Buen Trabajo, " + nombre + " adivinaste en ", adivina, " aportunidades")
if dato != numero:
        numero = str(numero)
        print("Ni modo, el numero que estaba pensando es ", numero)
```

El siguiente programa realiza la encripción / desencripción de una palabra utilizando el método de cifrado Cesar:

```
alfabeto = {"a":1,"b":2,"c":3,"d":4,"e":5,"f":6,"g":7,"h":8,"i":9,
        "j":10,"k":11,"l":12,"m":13,"n":14,"o":15,"p":16,"q":17,"r":18,
        "s":19,"t":20,"u":21,"v":22,"w":23,"x":24,"y":25,"z":26}
inverso = {1:"a",2:"b",3:"c",4:"d",5:"e",6:"f",7:"g",8:"h",9:"i",
        10:"j",11:"k",12:"l",13:"m",14:"n",15:"o",16:"p",17:"q",18:"r",
        19:"s",20:"t",21:"u",22:"v",23:"w",24:"x",25:"y",26:"z"}
ciclo = "S"
print ("Cifrado Cesar\n")
while ciclo == "S":
        contador = 0
        cambio = 0
        longitud = 0
        nueva_palabra = ""
        palabra_final = ""
        palabra = []
        letra = ()
        letras_palabra = []
        numeros = []
        cambio_numeros = []
        codificado = []
        b = ()
        menu = input("Selecciona Encriptar (1) Desencriptar (2): ")
        if menu == "1":
           palabra_final = "Codificado"
           palabra = input("Teclea la Palabra a Codificar: ")
        elif menu == "2" :
           palabra_final = "Decodificado"
           palabra = input("Teclea la Palabra a Decodificar: ")
        else:
           print ("Opcion Invalida. Vuelve a Intentarlo")
        letra = input ("Selecciona la letra en que la letra \"A\" se convertira: ")
        letra = (letra.lower())          # Convierte la letra a minusculas
        palabra = (palabra.lower())   # Convierte la palabra en minusculas
        longitud = len(palabra)
        cambio = alfabeto [letra] - 1
        while contador < longitud:
           letras_palabra.append(palabra [contador]) #put letters into file 'letras_palabra'
           numeros.append(int(alfabeto[palabra [contador]])) # Convierte las letras en numeros
```

```
            if menu == "1":
                cambio_numeros.append(numeros [contador] + cambio)
            if menu == "2":
                cambio_numeros.append(numeros [contador] - cambio)
            if cambio_numeros[contador] > 26:    # Ajusta los numeros en el alfabeto
                cambio_numeros[contador] -= 26
           if cambio_numeros[contador] < 1:
                cambio_numeros[contador] += 26
            codificado.append (inverso[cambio_numeros[contador]]) # coloca palabra condificada
            contador += 1
        contador = 0 # Concatena la palabra codificada
        while contador < longitud:
            b = codificado[contador]
            nueva_palabra = nueva_palabra + b
            contador += 1
        print("\nEl cambio de ",cambio," posiciones ",palabra_final," de la palabra ",palabra," es: ",nueva_palabra, " \n")
        ciclo = input("Deseas Continuar? (S / N): ").upper()
        print()
```

Para mayor información de cómo funciona el Cifrado Cesar ve a la siguiente liga:
http://es.wikipedia.org/wiki/Cifrado_C%C3%A9sar

APENDICE A – BIBLIOGRAFÍA

Los libros que recomiendo y que pueden ser un buen material de apoyo son los siguientes:

Python Programming: An Introduction to Computer Science 2nd Edition
Fecha de Edición: Mayo, 2010
Autores: John Zelle & Michael Smith
Idioma: Ingles
ISBN-10: 1590282418
ISBN-13: 978-1590282410

Python 3 for Absolute Beginners
Fecha de Edición: Octubre, 2009
Autores: Tim Hall & J-P Stacey
Idioma: Ingles
ISBN-10: 1430216328
ISBN-13: 978-1430216322

Python: Visual QuickStart Guide (3rd Edition)
Fecha de Edición: Julio 2013
Autor: Toby Donaldson
Idioma: Ingles
ISBN-10: 0321929551
ISBN-13: 978-0321929556

APENDICE B – RECURSOS

MATERIAL EXTERNO

A continuación se muestra una lista de enlaces que son material de apoyo en idioma español:
http://docs.python.org.ar/tutorial/pdfs/TutorialPython3.pdf
http://www.uji.es/bin/publ/edicions/ippython.pdf
http://python-para-impacientes.blogspot.com.es/p/indice.html
http://mundogeek.net/tutorial-python/

Unsted puede bajar el presente manual, así como su material de apoyo y futuras actualizaciones en la siguiente liga:

EL AUTOR

El presente manual fue elaborado por el Ingeniero **Carlos Vizcarra Lugo**, egresado del Instituto Tecnológico Autónomo de México (ITAM) de la carrera de Ingeniería en Computación.
Para cualquier pregunta o aclaración, el autor puede ser contactado en el correo electrónico siguiente: carvizcarra@yahoo.com

VERSIÓN

V0	Mayo 2014	Manual elaborado y en revisión su Contenido.
V1	Julio 2014	Documento revisado y liberado para su distribución.

LICENCIA