

The image features a red octagonal background, characteristic of the Python logo. The background is filled with faint, semi-transparent Python code snippets in a light red color. The code includes various elements such as class methods, file handling, and request processing. The word 'PYTHON' is prominently displayed in the center in a large, bold, white, sans-serif font.

PYTHON

www.postparaprogramadores.com

Contenido

Python - Inicio

Python - Descripción general

Python - Configuración del entorno

Python - Sintaxis Básica

Python - Tipos de variables

Python - Operadores básicos

Python - Toma de decisiones

Python - Bucles

Python - Números

Python - Cadenas

Python - Listas

Python - Tuplas

Python - Diccionario

Python - Fecha y hora

Python - Funciones

Python - Módulos

Python - E / S de archivos

Python - Excepciones

Tutorial avanzado de Python

Python - Clases / Objetos

Python - Expresiones Reg

Python - Programación CGI

Python - Acceso a la base de datos

Python - Redes

Python - Envío de correo electrónico

Python - Multithreading

Python - Procesamiento XML

Python - Programación GUI

Python - Extensiones adicionales

Descripción general de Python

Python es un lenguaje de scripting de alto nivel, interpretado, interactivo y orientado a objetos. Python está diseñado para ser altamente legible. Utiliza palabras clave en inglés con frecuencia, mientras que otros idiomas usan signos de puntuación, y tiene menos construcciones sintácticas que otros idiomas.

- **Python se interpreta** : Python es procesado en tiempo de ejecución por el intérprete. No necesita compilar su programa antes de ejecutarlo. Esto es similar a PERL y PHP.
- **Python es interactivo** : puede sentarse en un indicador de Python e interactuar con el intérprete directamente para escribir sus programas.
- **Python está orientado a objetos** : Python admite el estilo o la técnica de programación orientada a objetos que encapsula el código dentro de los objetos.
- **Python es un lenguaje para principiantes**: Python es un excelente lenguaje para los programadores de nivel principiante y admite el desarrollo de una amplia gama de aplicaciones, desde el procesamiento de texto simple hasta los navegadores WWW y los juegos.

Descarga más libros de programación GRATIS [click aquí](#)



Síguenos en Instagram para que estés al tanto de los nuevos libros de programación. [Click aquí](#)

Historia de Python

Python fue desarrollado por Guido van Rossum a fines de los años ochenta y principios de los noventa en el Instituto Nacional de Investigación de Matemáticas e Informática en los Países Bajos.

Python se deriva de muchos otros lenguajes, incluidos ABC, Modula-3, C, C++, Algol-68, SmallTalk y Unix shell y otros lenguajes de secuencias de comandos.

Python tiene derechos de autor. Al igual que Perl, el código fuente de Python ahora está disponible bajo la Licencia Pública General de GNU (GPL).

Python ahora es mantenido por un equipo de desarrollo central en el instituto, aunque Guido van Rossum todavía tiene un papel vital en la dirección de su progreso.

Características de Python

Las características de Python incluyen:

- **Fácil de aprender** : Python tiene pocas palabras clave, estructura simple y una sintaxis claramente definida. Esto le permite al alumno aprender el idioma rápidamente.
- **Fácil de leer** : el código de Python está más claramente definido y visible para los ojos.
- **Fácil de mantener** : el código fuente de Python es bastante fácil de mantener.
- **Una biblioteca estándar amplia** : la mayor parte de la biblioteca de Python es muy portátil y compatible con plataformas cruzadas en UNIX, Windows y Macintosh.
- **Modo interactivo** : Python admite un modo interactivo que permite realizar pruebas interactivas y depurar fragmentos de código.
- **Portátil** : Python puede ejecutarse en una amplia variedad de plataformas de hardware y tiene la misma interfaz en todas las plataformas.
- **Extensible** : puede agregar módulos de bajo nivel al intérprete de Python. Estos módulos permiten a los programadores agregar o personalizar sus herramientas para ser más eficientes.
- **Bases de datos** : Python proporciona interfaces a todas las principales bases de datos comerciales.
- **Programación GUI** : Python admite aplicaciones GUI que se pueden crear y portar a muchas llamadas de sistema, bibliotecas y sistemas Windows, como Windows MFC, Macintosh y el sistema X Window de Unix.
- **Escalable** : Python proporciona una mejor estructura y soporte para programas grandes que los scripts de shell.

Además de las características mencionadas anteriormente, Python tiene una gran lista de buenas características, algunas se enumeran a continuación:

- Admite métodos de programación funcionales y estructurados, así como OOP.
- Se puede usar como un lenguaje de scripting o se puede compilar en código de bytes para construir aplicaciones grandes.
- Proporciona tipos de datos dinámicos de muy alto nivel y admite la verificación de tipos dinámicos.
- Es compatible con la recolección automática de basura.
- Se puede integrar fácilmente con C, C ++, COM, ActiveX, CORBA y Java.

Python - Configuración del entorno

Python está disponible en una amplia variedad de plataformas, incluidas Linux y Mac OS X. Comprendamos cómo configurar nuestro entorno Python.

Configuración del entorno local

Abra una ventana de terminal y escriba "python" para averiguar si ya está instalado y qué versión está instalada.

- Unix (Solaris, Linux, FreeBSD, AIX, HP/UX, SunOS, IRIX, etc.)
- Win 9x/NT/2000
- Macintosh (Intel, PPC, 68K)
- OS/2
- DOS (multiple versions)
- PalmOS
- Nokia mobile phones
- Windows CE
- Acorn/RISC OS
- BeOS
- Amiga
- VMS/OpenVMS
- QNX
- VxWorks
- Psion

Conseguir Python

El código fuente, los binarios, la documentación, las noticias, etc. más actualizados y actuales están disponibles en el sitio web oficial de Python <https://www.python.org/>

Puede descargar la documentación de Python desde <https://www.python.org/doc/> . La documentación está disponible en formatos HTML, PDF y PostScript.

Instalando Python

La distribución de Python está disponible para una amplia variedad de plataformas. Necesita descargar solo el código binario aplicable a su plataforma e instalar Python.

Si el código binario para su plataforma no está disponible, necesita un compilador de C para compilar el código fuente manualmente. Compilar el código fuente ofrece más flexibilidad en cuanto a la elección de las características que necesita en su instalación.

Aquí hay una descripción rápida de la instalación de Python en varias plataformas:

Instalación de Unix y Linux

Estos son los pasos simples para instalar Python en la máquina Unix / Linux.

- Abra un navegador web y vaya a <https://www.python.org/downloads/> .
- Siga el enlace para descargar el código fuente comprimido disponible para Unix / Linux.
- Descargar y extraer archivos.
- Edición del archivo *Módulos / Configuración* si desea personalizar algunas opciones.
- ejecute `./configure script`
- hacer
- hacer instalar

Esto instala Python en la ubicación estándar `/usr / local / bin` y sus bibliotecas en `/usr / local / lib / pythonXX` donde XX es la versión de Python.

Instalación de ventanas

Estos son los pasos para instalar Python en una máquina con Windows.

- Abra un navegador web y vaya a <https://www.python.org/downloads/> .
- Siga el enlace para el archivo de instalación de Windows *python-XYZ.msi* donde XYZ es la versión que necesita instalar.
- Para usar este instalador *python-XYZ.msi* , el sistema Windows debe ser compatible con Microsoft Installer 2.0. Guarde el archivo de instalación en su máquina local y luego ejecútelo para averiguar si su máquina es compatible con MSI.
- Ejecute el archivo descargado. Esto muestra el asistente de instalación de Python, que es realmente fácil de usar. Simplemente acepte la configuración predeterminada, espere hasta que finalice la instalación y haya terminado.

Instalación de Macintosh

Macs recientes vienen con Python instalado, pero puede estar desactualizado varios años. Consulte <http://www.python.org/download/mac/> para obtener instrucciones sobre cómo obtener la versión actual junto con herramientas adicionales para apoyar el desarrollo en Mac. Para Mac OS anteriores a Mac OS X 10.3 (lanzado en 2003), MacPython está disponible.

Jack Jansen lo mantiene y puede tener acceso completo a toda la documentación en su sitio web: <http://www.cwi.nl/~jack/macpython.html> . Puede encontrar detalles completos de instalación para la instalación de Mac OS.

Configurar RUTA

Los programas y otros archivos ejecutables pueden estar en muchos directorios, por lo que los sistemas operativos proporcionan una ruta de búsqueda que enumera los directorios que el sistema operativo busca ejecutables.

La ruta se almacena en una variable de entorno, que es una cadena con nombre mantenida por el sistema operativo. Esta variable contiene información disponible para el shell de comandos y otros programas.

La variable de **ruta** se denomina RUTA en Unix o Ruta en Windows (Unix distingue entre mayúsculas y minúsculas; Windows no).

En Mac OS, el instalador maneja los detalles de la ruta. Para invocar al intérprete de Python desde cualquier directorio en particular, debe agregar el directorio de Python a su ruta.

Establecer ruta en Unix / Linux

Para agregar el directorio de Python a la ruta de una sesión en particular en Unix:

- **En el shell csh** , escriba `setenv PATH "$ PATH: / usr / local / bin / python"` y presione Entrar.
- **En el shell bash (Linux)** , escriba `export PATH = "$ PATH: / usr / local / bin / python"` y presione Entrar.
- **En el shell sh o ksh** , escriba `PATH = "$ PATH: / usr / local / bin / python"` y presione Entrar.
- **Nota** : / usr / local / bin / python es la ruta del directorio de Python

Establecer ruta en Windows

Para agregar el directorio de Python a la ruta de una sesión en particular en Windows:

En el símbolo del sistema , escriba `path% path%; C: \ Python` y presione Entrar.

Nota : C: \ Python es la ruta del directorio de Python

Variables de entorno de Python

Aquí hay importantes variables de entorno, que Python puede reconocer:

No.	Variable y descripción
1	RUTA DEL PITÓN Tiene un papel similar al de la RUTA. Esta variable le dice al intérprete de Python dónde ubicar los archivos del módulo importados a un programa. Debe incluir el directorio de la biblioteca fuente de Python y los directorios que contienen el código fuente de Python. PYTHONPATH a veces está preestablecido por el instalador de Python.
2	PYTHONSTARTUP Contiene la ruta de un archivo de inicialización que contiene el código fuente de Python. Se ejecuta cada vez que inicia el intérprete. Se nombra como .pythonrc.py en Unix y contiene comandos que cargan utilidades o modifican PYTHONPATH.
3	PYTHONCASEOK Se usa en Windows para indicar a Python que busque la primera coincidencia entre mayúsculas y minúsculas en una declaración de importación. Establezca esta variable en cualquier valor para activarla.

4 4

PYTHONHOME

Es una ruta de búsqueda de módulo alternativa. Por lo general, está incrustado en los directorios PYTHONSTARTUP o PYTHONPATH para facilitar el cambio de bibliotecas de módulos.

Corriendo Python

Hay tres formas diferentes de iniciar Python:

Intérprete interactivo

Puede iniciar Python desde Unix, DOS o cualquier otro sistema que le proporcione un intérprete de línea de comandos o una ventana de shell.

Ingrese **python** en la línea de comando.

Comience a codificar de inmediato en el intérprete interactivo.

```
$python # Unix/Linux  
or  
python% # Unix/Linux  
or  
C:> python # Windows/DOS
```

Aquí está la lista de todas las opciones de línea de comando disponibles:

No.	Opción y descripción
1	-re Proporciona resultados de depuración.
2	-O Genera bytecode optimizado (que resulta en archivos .pyo).
3	-S No ejecute el sitio de importación para buscar rutas de Python en el inicio.
4 4	-v salida detallada (seguimiento detallado en las declaraciones de importación).
5 5	-X deshabilitar excepciones incorporadas basadas en clases (solo use cadenas); obsoleto a partir de la versión 1.6.

6 6	-c cmd ejecutar el script Python enviado como cadena cmd
7 7	archivo ejecutar script de Python desde un archivo dado

Script desde la línea de comando

Un script de Python se puede ejecutar en la línea de comandos invocando al intérprete en su aplicación, como se muestra a continuación:

```
$python script.py # Unix/Linux
```

or

```
python% script.py # Unix/Linux
```

or

```
C: >python script.py # Windows/DOS
```

Nota : asegúrese de que el modo de permiso de archivo permita la ejecución.

Entorno de desarrollo integrado

También puede ejecutar Python desde un entorno de interfaz gráfica de usuario (GUI), si tiene una aplicación GUI en su sistema que admita Python.

- **Unix** : IDLE es el primer IDE de Unix para Python.
- **Windows** : PythonWin es la primera interfaz de Windows para Python y es un IDE con una GUI.
- **Macintosh** : la versión Macintosh de Python junto con IDLE IDE está disponible en el sitio web principal, descargable como archivos MacBinary o BinHex'd.

Si no puede configurar el entorno correctamente, puede pedir ayuda al administrador del sistema. Asegúrese de que el entorno de Python esté configurado correctamente y funcione perfectamente bien.

Nota : Todos los ejemplos dados en capítulos posteriores se ejecutan con la versión Python 2.4.3 disponible en CentOS sabor de Linux.

Ya hemos configurado el entorno de programación Python en línea, para que pueda ejecutar todos los ejemplos disponibles en línea al mismo tiempo cuando esté aprendiendo teoría. Siéntase libre de modificar cualquier ejemplo y ejecutarlo en línea.

Python - Sintaxis Básica

El lenguaje Python tiene muchas similitudes con Perl, C y Java. Sin embargo, hay algunas diferencias definitivas entre los idiomas.

Primer programa de Python

Ejecutemos programas en diferentes modos de programación.

Programación en modo interactivo

Invocar al intérprete sin pasar un archivo de script como parámetro muestra el siguiente mensaje:

```
$ python
Python 2.4.3 (#1, Nov 11 2010, 13:34:43)
[GCC 4.1.2 20080704 (Red Hat 4.1.2-48)] on linux2
Type "help", "copyright", "credits" or "license" for more
information.
>>>
```

Escriba el siguiente texto en el indicador de Python y presione Entrar -

```
>>> print "Hello, Python!"
```

Si está ejecutando una nueva versión de Python, entonces necesitaría usar la declaración de impresión con paréntesis como en **print ("¡Hola, Python!");**. Sin embargo, en Python versión 2.4.3, esto produce el siguiente resultado:

```
Hello, Python!
```

Programación en modo script

Invocar al intérprete con un parámetro de script comienza la ejecución del script y continúa hasta que finaliza el script. Cuando finaliza el guión, el intérprete ya no está activo.

Escribamos un programa simple de Python en un script. Los archivos de Python tienen la extensión **.py**. Escriba el siguiente código fuente en un archivo test.py:

```
print "Hello, Python!"
```

Suponemos que tiene un intérprete de Python establecido en la variable PATH. Ahora, intente ejecutar este programa de la siguiente manera:

```
$ python test.py
```

Esto produce el siguiente resultado:

```
Hello, Python!
```

Probemos otra forma de ejecutar un script Python. Aquí está el archivo modificado test.py:

```
#!/usr/bin/python  
  
print "Hello, Python!"
```

Asumimos que tiene un intérprete de Python disponible en el directorio /usr/bin. Ahora, intente ejecutar este programa de la siguiente manera:

```
$ chmod +x test.py      # This is to make file executable  
$ ./test.py
```

Esto produce el siguiente resultado:

```
Hello, Python!
```

Identificadores de Python

Un identificador de Python es un nombre utilizado para identificar una variable, función, clase, módulo u otro objeto. Un identificador comienza con una letra A a Z o una a z o un guión bajo (_) seguido de cero o más letras, guiones bajos y dígitos (0 a 9).

Python no permite caracteres de puntuación como @, \$ y % dentro de los identificadores. Python es un lenguaje de programación sensible a mayúsculas y minúsculas. Por lo tanto, **Manpower** y **manpower** son dos identificadores diferentes en Python.

Aquí hay convenciones de nombres para identificadores de Python:

- Los nombres de clase comienzan con una letra mayúscula. Todos los demás identificadores comienzan con una letra minúscula.
- Iniciar un identificador con un solo guión bajo indica que el identificador es privado.
- Comenzar un identificador con dos guiones bajos indica un identificador fuertemente privado.
- Si el identificador también termina con dos guiones bajos, el identificador es un nombre especial definido por el idioma.

Descarga más libros de programación GRATIS [click aquí](#)

Palabras reservadas

La siguiente lista muestra las palabras clave de Python. Estas son palabras reservadas y no puede usarlas como constantes o variables ni ningún otro nombre identificador. Todas las palabras clave de Python contienen solo letras minúsculas.

and	exec	not
assert	finally	or
break	for	pass
class	from	print
continue	global	raise
def	if	return
del	import	try
elif	in	while
else	is	with
except	lambda	yield

Líneas y sangría

Python no proporciona llaves para indicar bloques de código para las definiciones de clase y función o control de flujo. Los bloques de código se denotan mediante sangría de línea, que se aplica de forma rígida.

El número de espacios en la sangría es variable, pero todas las declaraciones dentro del bloque deben sangrar la misma cantidad. Por ejemplo

```
if True:
    print "True"
else:
    print "False"
```

Sin embargo, el siguiente bloque genera un error:

```
if True:
print "Answer"
print "True"
else:
print "Answer"
print "False"
```

Por lo tanto, en Python todas las líneas continuas sangradas con el mismo número de espacios formarían un bloque. El siguiente ejemplo tiene varios bloques de instrucciones:

Nota : no intente comprender la lógica en este momento. Solo asegúrese de comprender varios bloques, incluso si no tienen llaves.

```
#!/usr/bin/python

import sys

try:
    # open file stream
    file = open(file_name, "w")
except IOError:
    print "There was an error writing to", file_name
    sys.exit()

print "Enter '", file_finish,
print "' When finished"
while file_text != file_finish:
    file_text = raw_input("Enter text: ")
    if file_text == file_finish:
        # close the file
        file.close
        break
    file.write(file_text)
    file.write("\n")
file.close()
file_name = raw_input("Enter filename: ")
if len(file_name) == 0:
    print "Next time please enter something"
    sys.exit()
```



```
try:
    file = open(file_name, "r")
except IOError:
    print "There was an error reading file"
    sys.exit()
file_text = file.read()
file.close()
print file_text
```

Declaraciones de líneas múltiples

Las declaraciones en Python generalmente terminan con una nueva línea. Python, sin embargo, permite el uso del carácter de continuación de línea (\) para indicar que la línea debe continuar. Por ejemplo

```
total = item_one + \
        item_two + \
        item_three
```

Las declaraciones contenidas dentro de los corchetes [], {} o () no necesitan usar el carácter de continuación de línea. Por ejemplo

```
days = ['Monday', 'Tuesday', 'Wednesday',
        'Thursday', 'Friday']
```

Cita en Python

Python acepta comillas simples ('), dobles (") y triples ('" o ""') para denotar literales de cadena, siempre que el mismo tipo de comilla comience y termine la cadena.

Las comillas triples se utilizan para abarcar la cadena a través de varias líneas. Por ejemplo, todo lo siguiente es legal:

```
word = 'word'
sentence = "This is a sentence."
paragraph = """This is a paragraph. It is
made up of multiple lines and sentences."""
```

Comentarios en Python

Un signo hash (#) que no está dentro de un literal de cadena comienza un comentario. Todos los caracteres después del # y hasta el final de la línea física son parte del comentario y el intérprete de Python los ignora.

```
#!/usr/bin/python
# First comment
print "Hello, Python!" # second comment
```

Esto produce el siguiente resultado:

```
Hello, Python!
```

Puede escribir un comentario en la misma línea después de una declaración o expresión:

```
name = "Madisetti" # This is again comment
```

Puede comentar varias líneas de la siguiente manera:

```
# This is a comment.  
# This is a comment, too.  
# This is a comment, too.  
# I said that already.
```

Usando líneas en blanco

Una línea que contiene solo espacios en blanco, posiblemente con un comentario, se conoce como una línea en blanco y Python la ignora por completo.

En una sesión de intérprete interactiva, debe ingresar una línea física vacía para terminar una instrucción multilínea.

Esperando al usuario

La siguiente línea del programa muestra el mensaje, la declaración que dice "Presione la tecla Intro para salir", y espera a que el usuario tome medidas:

```
#!/usr/bin/python  
  
raw_input("\n\nPress the enter key to exit.")
```

Aquí, "\ n \ n" se usa para crear dos líneas nuevas antes de mostrar la línea real. Una vez que el usuario presiona la tecla, el programa finaliza. Este es un buen truco para mantener una ventana de consola abierta hasta que el usuario haya terminado con una aplicación.

Múltiples declaraciones en una sola línea

El punto y coma (;) permite múltiples sentencias en la única línea dado que ninguna de las sentencias inicia un nuevo bloque de código. Aquí hay un recorte de muestra usando el punto y coma:

```
import sys; x = 'foo'; sys.stdout.write(x + '\n')
```

Descarga más libros de programación GRATIS [click aquí](#)

Grupos de estados de cuenta múltiples como suites

Un grupo de declaraciones individuales, que forman un solo bloque de código, se denominan **suites** en Python. Las declaraciones compuestas o complejas, como `if`, `while`, `def` y `class` requieren una línea de encabezado y una suite.

Las líneas de encabezado comienzan la declaración (con la palabra clave) y terminan con dos puntos (`:`) y son seguidas por una o más líneas que conforman el conjunto. Por ejemplo

```
if expression :
    suite
elif expression :
    suite
else :
    suite
```

Argumentos de línea de comando

Se pueden ejecutar muchos programas para proporcionarle información básica sobre cómo deben ejecutarse. Python le permite hacer esto con `-h` -

```
$ python -h
usage: python [option] ... [-c cmd | -m mod | file | -] [arg]
...
Options and arguments (and corresponding environment
variables):
-c cmd : program passed in as string (terminates option list)
-d      : debug output from parser (also PYTHONDEBUG=x)
-E      : ignore environment variables (such as PYTHONPATH)
-h      : print this help message and exit

[ etc. ]
```

También puede programar su script de tal manera que acepte varias opciones. [Command Line Arguments](#) es un tema avanzado y debe estudiarse un poco más tarde una vez que haya revisado el resto de los conceptos de Python.

Python - Tipos de variables

Las variables no son más que ubicaciones de memoria reservadas para almacenar valores. Esto significa que cuando crea una variable, reserva algo de espacio en la memoria.

Según el tipo de datos de una variable, el intérprete asigna memoria y decide qué se puede almacenar en la memoria reservada. Por lo tanto, al asignar diferentes tipos de datos a las variables, puede almacenar enteros, decimales o caracteres en estas variables.

Asignación de valores a variables

Las variables de Python no necesitan una declaración explícita para reservar espacio en la memoria. La declaración ocurre automáticamente cuando asigna un valor a una variable. El signo igual (=) se usa para asignar valores a las variables.

El operando a la izquierda del operador = es el nombre de la variable y el operando a la derecha del operador = es el valor almacenado en la variable. Por ejemplo

```
#!/usr/bin/python
```

```
counter = 100           # An integer assignment  
miles   = 1000.0       # A floating point  
name    = "John"       # A string
```

```
print counter  
print miles  
print name
```

Aquí, 100, 1000.0 y "John" son los valores asignados a las variables *contador*, *millas* y *nombre*, respectivamente. Esto produce el siguiente resultado:

```
100  
1000.0  
John
```

Asignación Múltiple

Python le permite asignar un solo valor a varias variables simultáneamente. Por ejemplo

```
a = b = c = 1
```

Aquí, se crea un objeto entero con el valor 1, y las tres variables se asignan a la misma ubicación de memoria. También puede asignar múltiples objetos a múltiples variables. Por ejemplo

```
a,b,c = 1,2,"john"
```

Aquí, dos objetos enteros con valores 1 y 2 se asignan a las variables a y b respectivamente, y un objeto de cadena con el valor "john" se asigna a la variable c.

Tipos de datos estándar

Los datos almacenados en la memoria pueden ser de muchos tipos. Por ejemplo, la edad de una persona se almacena como un valor numérico y su dirección se almacena como caracteres alfanuméricos. Python tiene varios tipos de datos estándar que se utilizan para definir las operaciones posibles en ellos y el método de almacenamiento para cada uno de ellos.

Python tiene cinco tipos de datos estándar:

- Números
- Cuerda
- Lista
- Tupla
- Diccionario

Números de Python

Los tipos de datos numéricos almacenan valores numéricos. Los objetos numéricos se crean cuando les asigna un valor. Por ejemplo

```
var1 = 1
var2 = 10
```

También puede eliminar la referencia a un objeto numérico utilizando la instrucción del. La sintaxis de la declaración del es -

```
del var1[,var2[,var3[....,varN]]]
```

Puede eliminar un solo objeto o varios objetos utilizando la instrucción del. Por ejemplo

```
del var
del var_a, var_b
```

Python admite cuatro tipos numéricos diferentes:

- int (enteros con signo)
- long (enteros largos, también se pueden representar en octal y hexadecimal)
- flotante (valores reales de coma flotante)
- complejo (números complejos)

Ejemplos

Aquí hay algunos ejemplos de números:

En t	largo	flotador	complejo
10	51924361L	0.0	3.14j
100	-0x19323L	15,20	45.j
-786	0122L	-21,9	9.322e-36j
080	0xDEFABCECBDAECBFBAEI	32,3 + e18	.876j
-0490	535633629843L	-90.	-.6545 + 0J
-0x260	-052318172735L	-32.54e100	3e + 26J
0x69	-4721885298529L	70.2-E12	4.53e-7j

- Python le permite usar una l minúscula con largo, pero se recomienda que use solo una L mayúscula para evitar confusión con el número 1. Python muestra enteros largos con una L mayúscula.
- Un número complejo consiste en un par ordenado de números de coma flotante reales denotados por $x + yj$, donde x e y son los números reales y j es la unidad imaginaria.

Cadenas o String

Las cadenas en Python se identifican como un conjunto contiguo de caracteres representados en las comillas. Python permite pares de comillas simples o dobles. Se pueden tomar subconjuntos de cadenas usando el operador de división ([] y [:]) con índices que comienzan en 0 al comienzo de la cadena y van desde -1 al final.

El signo más (+) es el operador de concatenación de cadenas y el asterisco (*) es el operador de repetición. Por ejemplo

```
#!/usr/bin/python

str = 'Hello World!'

print str           # Prints complete string
print str[0]       # Prints first character of the string
print str[2:5]     # Prints characters starting from 3rd to
5th
print str[2:]      # Prints string starting from 3rd
character
print str * 2      # Prints string two times
print str + "TEST" # Prints concatenated string
```

Esto producirá el siguiente resultado:

```
Hello World!
H
llo
llo World!
Hello World!Hello World!
Hello World!TEST
```

Listas de Python

Las listas son los tipos de datos compuestos más versátiles de Python. Una lista contiene elementos separados por comas y encerrados entre corchetes ([]). Hasta cierto punto, las listas son similares a las matrices en C. Una diferencia entre ellas es que todos los elementos que pertenecen a una lista pueden ser de diferentes tipos de datos.

Se puede acceder a los valores almacenados en una lista utilizando el operador de división ([] y [:]) con índices que comienzan en 0 al comienzo de la lista y se abren camino hasta el final -1. El signo más (+) es el operador de concatenación de lista, y el asterisco (*) es el operador de repetición. Por ejemplo

```
#!/usr/bin/python

list = [ 'abcd', 786 , 2.23, 'john', 70.2 ]
tinylist = ['john']
```

```

print list          # Prints complete list
print list[0]      # Prints first element of the list
print list[1:3]    # Prints elements starting from 2nd till
3rd
print list[2:]     # Prints elements starting from 3rd
element
print tinylist * 2 # Prints list two times
print list + tinylist # Prints concatenated lists

```

Esto produce el siguiente resultado:

```

['abcd', 786, 2.23, 'john', 70.2]
abcd
[786, 2.23]
[2.23, 'john', 70.2]
[123, 'john', 123, 'john']
['abcd', 786, 2.23, 'john', 70.2, 123, 'john']

```

Tuplas

Una tupla es otro tipo de datos de secuencia que es similar a la lista. Una tupla consiste en una serie de valores separados por comas. Sin embargo, a diferencia de las listas, las tuplas están encerradas entre paréntesis.

Las principales diferencias entre las listas y las tuplas son: las listas están entre corchetes ([]) y sus elementos y tamaño pueden cambiarse, mientras que las tuplas están entre paréntesis (()) y no pueden actualizarse. Las tuplas pueden considerarse como listas de **solo lectura** . Por ejemplo

```

#!/usr/bin/python

tuple = ( 'abcd', 786 , 2.23, 'john', 70.2 )
tinytuple = (123, 'john')

print tuple          # Prints complete list
print tuple[0]      # Prints first element of the list
print tuple[1:3]    # Prints elements starting from 2nd
till 3rd
print tuple[2:]     # Prints elements starting from 3rd
element
print tinytuple * 2 # Prints list two times
print tuple + tinytuple # Prints concatenated lists

```

Esto produce el siguiente resultado:

```

('abcd', 786, 2.23, 'john', 70.2)
abcd
(786, 2.23)
(2.23, 'john', 70.2)
(123, 'john', 123, 'john')
('abcd', 786, 2.23, 'john', 70.2, 123, 'john')

```


El siguiente código no es válido con tupla, porque intentamos actualizar una tupla, lo cual no está permitido. Caso similar es posible con listas -

```
#!/usr/bin/python

tuple = ( 'abcd', 786 , 2.23, 'john', 70.2 )
list = [ 'abcd', 786 , 2.23, 'john', 70.2 ]
tuple[2] = 1000      # Invalid syntax with tuple
list[2] = 1000      # Valid syntax with list
```

Diccionario Python

Los diccionarios de Python son un tipo de tabla hash. Funcionan como matrices asociativas o hashes que se encuentran en Perl y consisten en pares clave-valor. Una clave de diccionario puede ser casi cualquier tipo de Python, pero generalmente son números o cadenas. Los valores, por otro lado, pueden ser cualquier objeto arbitrario de Python.

Los diccionarios están encerrados entre llaves ({}), y los valores se pueden asignar y acceder usando llaves cuadradas ([]). Por ejemplo

```
#!/usr/bin/python

dict = {}
dict['one'] = "This is one"
dict[2]      = "This is two"

tinydict = {'name': 'john', 'code':6734, 'dept': 'sales'}

print dict['one']      # Prints value for 'one' key
print dict[2]          # Prints value for 2 key
print tinydict         # Prints complete dictionary
print tinydict.keys() # Prints all the keys
print tinydict.values() # Prints all the values
```

Esto produce el siguiente resultado:

```
This is one
This is two
{'dept': 'sales', 'code': 6734, 'name': 'john'}
['dept', 'code', 'name']
['sales', 6734, 'john']
```

Los diccionarios no tienen un concepto de orden entre los elementos. Es incorrecto decir que los elementos están "fuera de servicio"; simplemente no están ordenados.

Conversión de tipo de datos

A veces, es posible que deba realizar conversiones entre los tipos integrados. Para convertir entre tipos, simplemente use el nombre del tipo como una función.

Hay varias funciones integradas para realizar la conversión de un tipo de datos a otro. Estas funciones devuelven un nuevo objeto que representa el valor convertido.

No Señor.	Función descriptiva
1	int (x [, base]) Convierte x en un entero. base especifica la base si x es una cadena.
2	largo (x [, base]) Convierte x en un entero largo. base especifica la base si x es una cadena.
3	flotador (x) Convierte x en un número de coma flotante.
4 4	complejo (real [, imag]) Crea un número complejo.
5 5	str (x) Convierte el objeto x en una representación de cadena.
6 6	repr (x) Convierte el objeto x en una cadena de expresión.
7 7	eval (str) Evalúa una cadena y devuelve un objeto.
8	tupla (s) Convierte s en una tupla.
9 9	liza) Convierte s en una lista.

10	conjunto (s) Convierte s en un conjunto.
11	dict (d) Crea un diccionario. d debe ser una secuencia de tuplas (clave, valor).
12	conjuntos congelados Convierte s en un conjunto congelado.
13	chr (x) Convierte un entero en un carácter.
14	unichr (x) Convierte un entero en un carácter Unicode.
15	ord (x) Convierte un solo carácter a su valor entero.
dieciséis	hex (x) Convierte un entero en una cadena hexadecimal.
17	oct (x) Convierte un entero en una cadena octal.

Python - Operadores básicos

Los operadores son las construcciones que pueden manipular el valor de los operandos.

Considere la expresión $4 + 5 = 9$. Aquí, 4 y 5 se llaman operandos y + se llama operador.

Tipos de operador

El lenguaje Python admite los siguientes tipos de operadores.

- Operadores aritméticos
- Operadores de comparación (relacionales)
- Operadores de Asignación
- Operadores lógicos
- Operadores bit a bit
- Operadores de membresía
- Operadores de identidad

Echemos un vistazo a todos los operadores uno por uno.

Operadores aritméticos de Python

Suponga que la variable a tiene 10 y la variable b tiene 20, entonces -

[[Mostrar ejemplo](#)]

Operador	Descripción	Ejemplo
+ Adición	Agrega valores a ambos lados del operador.	$a + b = 30$
- Resta	Resta el operando de la derecha del operando de la izquierda.	$a - b = -10$
* Multiplicación	Multiplica los valores a cada lado del operador	$a * b = 200$
/ División	Divide el operando de la izquierda por el operando de la derecha	$b / a = 2$
% Módulo	Divide el operando de la izquierda por el operando de la derecha y devuelve el resto	$b \% a = 0$

** Exponente	Realiza cálculo exponencial (potencia) en operadores	a ** b = 10 a la potencia 20
//	División de piso: la división de operandos donde el resultado es el cociente en el que se eliminan los dígitos después del punto decimal. Pero si uno de los operandos es negativo, el resultado se anula, es decir, se redondea desde cero (hacia el infinito negativo):	9 // 2 = 4 y 9.0 // 2.0 = 4.0, -11 // 3 = -4, -11.0 // 3 = -4.0

Operadores de comparación de Python

Estos operadores comparan los valores a ambos lados de ellos y deciden la relación entre ellos. También se llaman operadores relacionales.

Suponga que la variable a tiene 10 y la variable b tiene 20, entonces -

[[Mostrar ejemplo](#)]

Operador	Descripción	Ejemplo
==	Si los valores de dos operandos son iguales, entonces la condición se vuelve verdadera.	(a == b) no es cierto.
!=	Si los valores de dos operandos no son iguales, entonces la condición se vuelve verdadera.	(a != b) es cierto.
<>	Si los valores de dos operandos no son iguales, entonces la condición se vuelve verdadera.	(a <> b) es cierto. Esto es similar a != Operador.
>	Si el valor del operando izquierdo es mayor que el valor del operando derecho, entonces la condición se vuelve verdadera.	(a > b) no es cierto.
<	Si el valor del operando izquierdo es menor que el valor del operando derecho, entonces la condición se vuelve verdadera.	(a < b) es cierto.
>=	Si el valor del operando izquierdo es mayor o igual que el valor del	(a >= b) no

	operando derecho, entonces la condición se vuelve verdadera.	es cierto.
<=	Si el valor del operando izquierdo es menor o igual que el valor del operando derecho, entonces la condición se vuelve verdadera.	(a <= b) es cierto.

Operadores de asignación de Python

Suponga que la variable a tiene 10 y la variable b tiene 20, entonces -

[[Mostrar ejemplo](#)]

Operador	Descripción	Ejemplo
=	Asigna valores de operandos del lado derecho al operando del lado izquierdo	c = a + b asigna el valor de a + b a c
+ = Agregar Y	Agrega el operando derecho al operando izquierdo y asigna el resultado al operando izquierdo	c + = a es equivalente a c = c + a
- = Restar Y	Resta el operando derecho del operando izquierdo y asigna el resultado al operando izquierdo	c - = a es equivalente a c = c - a
* = Multiplicar Y	Multiplica el operando derecho con el operando izquierdo y asigna el resultado al operando izquierdo	c * = a es equivalente a c = c * a
/ = Dividir Y	Divide el operando izquierdo con el operando derecho y asigna el resultado al operando izquierdo	c / = a es equivalente a c = c / a
% = Módulo Y	Toma módulo usando dos operandos y asigna el resultado al operando izquierdo	c% = a es equivalente a c = c% a
** = exponente Y	Realiza cálculos exponenciales (potencia) en operadores y asigna valor al operando izquierdo	c ** = a es equivalente a c = c ** a

// = División de piso	Realiza división de piso en operadores y asigna valor al operando izquierdo	c // a es equivalente a c = c // a
-----------------------	---	------------------------------------

Operadores bit a bit de Python

El operador bit a bit trabaja en bits y realiza la operación bit a bit. Supongamos que $a = 60$; $b = 13$; Ahora en formato binario sus valores serán 0011 1100 y 0000 1101 respectivamente. La siguiente tabla enumera los operadores bit a bit admitidos por el lenguaje Python con un ejemplo de cada uno de ellos, utilizamos las dos variables anteriores (a y b) como operandos:

a = 0011 1100

b = 0000 1101

a & b = 0000 1100

a | b = 0011 1101

a ^ b = 0011 0001

~ a = 1100 0011

Existen los siguientes operadores Bitwise compatibles con el lenguaje Python

[[Mostrar ejemplo](#)]

Operador	Descripción	Ejemplo
& Binary AND	El operador copia un poco al resultado si existe en ambos operandos	(a & b) (significa 0000 1100)
El O binario	Se copia un poco si existe en cualquiera de los operandos.	(a b) = 61 (significa 0011 1101)
^ Binario XOR	Copia el bit si se establece en un operando pero no en ambos.	(a ^ b) = 49 (significa 0011 0001)
~ Complemento de binarios	Es unario y tiene el efecto de 'voltear' bits.	(~ a) = -61 (significa 1100 0011 en forma de complemento a 2 debido a

		un número binario con signo.
<< Desplazamiento binario a la izquierda	El valor de los operandos de la izquierda se mueve hacia la izquierda por la cantidad de bits especificados por el operando de la derecha.	$a \ll 2 = 240$ (significa 1111 0000)
>> Desplazamiento binario a la derecha	El valor de los operandos de la izquierda se mueve hacia la derecha por la cantidad de bits especificados por el operando de la derecha.	$a \gg 2 = 15$ (significa 0000 1111)

Operadores lógicos de Python

Existen los siguientes operadores lógicos compatibles con el lenguaje Python. Suponga que la variable a tiene 10 y la variable b tiene 20 entonces

[[Mostrar ejemplo](#)]

Operador	Descripción	Ejemplo
y lógico Y	Si ambos operandos son verdaderos, entonces la condición se vuelve verdadera.	(ayb) es cierto.
o lógico O	Si alguno de los dos operandos no es cero, la condición se vuelve verdadera.	(aob) es cierto.
no lógico NO	Se usa para invertir el estado lógico de su operando.	No (a y b) es falso.

Operadores de membresía de Python

Los operadores de membresía de Python prueban la membresía en una secuencia, como cadenas, listas o tuplas. Hay dos operadores de membresía como se explica a continuación:

[[Mostrar ejemplo](#)]

Operador	Descripción	Ejemplo
en	Evalúa a verdadero si encuentra una variable en la secuencia especificada y falso en caso contrario.	x en y, aquí resulta en un 1 si x es un miembro de la secuencia y.
no en	Evalúa a verdadero si no encuentra una variable en la secuencia especificada y falso en caso contrario.	x no en y, aquí no en resultados en un 1 si x no es miembro de la secuencia y.

Operadores de identidad de Python

Los operadores de identidad comparan las ubicaciones de memoria de dos objetos. A continuación se explican dos operadores de identidad:

[[Mostrar ejemplo](#)]

Operador	Descripción	Ejemplo
es	Evalúa a verdadero si las variables a cada lado del operador apuntan al mismo objeto y falso de lo contrario.	x es y, aquí hay resultados en 1 si id (x) es igual a id (y).
no es	Evalúa a falso si las variables a cada lado del operador	x no es y, aquí no hay resultados en 1 si id (x) no es igual

	apuntan al mismo objeto y verdadero de lo contrario.	a id (y).
--	--	-----------

Precedencia de operadores de Python

La siguiente tabla enumera todos los operadores desde la prioridad más alta hasta la más baja.

[[Mostrar ejemplo](#)]

No Señor.	Operador y Descripción
1	** ** Exponenciación (elevar al poder)
2	~ + - Complemento, unario más y menos (los nombres de método para los dos últimos son + @ y - @)
3	* /% // División de multiplicación, división, módulo y piso
4 4	+ - Adición y sustracción
5 5	>> << Desplazamiento bit a la derecha e izquierda
6 6	Y Bitwise 'Y'
7 7	^ Bitwise exclusivo 'OR' y regular 'OR'
8	<= <> = Operadores de comparación

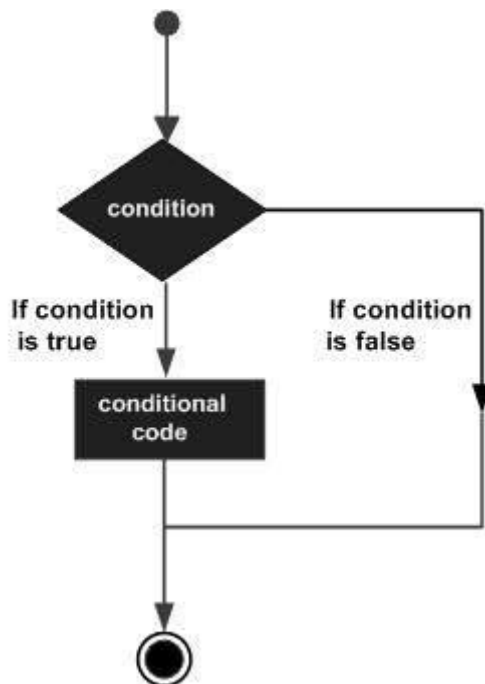
9 9	<> ==! = Operadores de igualdad
10	=% = / = // = - = + = * = ** = Operadores de Asignación
11	no lo es Operadores de identidad
12	en no en Operadores de membresía
13	no o y Operadores logicos

Python - Toma de decisiones

La toma de decisiones es la anticipación de las condiciones que ocurren mientras se ejecuta el programa y se especifican las acciones tomadas de acuerdo con las condiciones.

Las estructuras de decisión evalúan múltiples expresiones que producen VERDADERO o FALSO como resultado. Debe determinar qué acción tomar y qué declaraciones ejecutar si el resultado es VERDADERO o FALSO de lo contrario.

A continuación se presenta la forma general de una estructura de toma de decisiones típica que se encuentra en la mayoría de los lenguajes de programación:



El lenguaje de programación Python asume cualquier valor **distinto de cero** y **no nulo** como VERDADERO, y si es **cero** o **nulo**, se asume como valor FALSO.

El lenguaje de programación Python proporciona los siguientes tipos de declaraciones de toma de decisiones. Haga clic en los siguientes enlaces para verificar sus detalles.

No Señor.	Declaración y descripción
1	si declaraciones Una declaración if consiste en una expresión booleana seguida de una o más declaraciones.
2	si ... otras declaraciones

	Una instrucción if puede ser seguida por una instrucción else opcional , que se ejecuta cuando la expresión booleana es FALSE.
3	<u>instrucciones if anidadas</u> Puede usar una declaración if o else if dentro de otra declaración if o else if (s) .

Veamos brevemente cada toma de decisiones:

Suites de declaración única

Si el conjunto de una cláusula **if** consta solo de una línea, puede ir en la misma línea que la instrucción de encabezado.

Aquí hay un ejemplo de una cláusula **if de una línea** :

```
#!/usr/bin/python  
  
var = 100  
if ( var == 100 ) : print "Value of expression is 100"  
print "Good bye!"
```

Cuando se ejecuta el código anterior, produce el siguiente resultado:

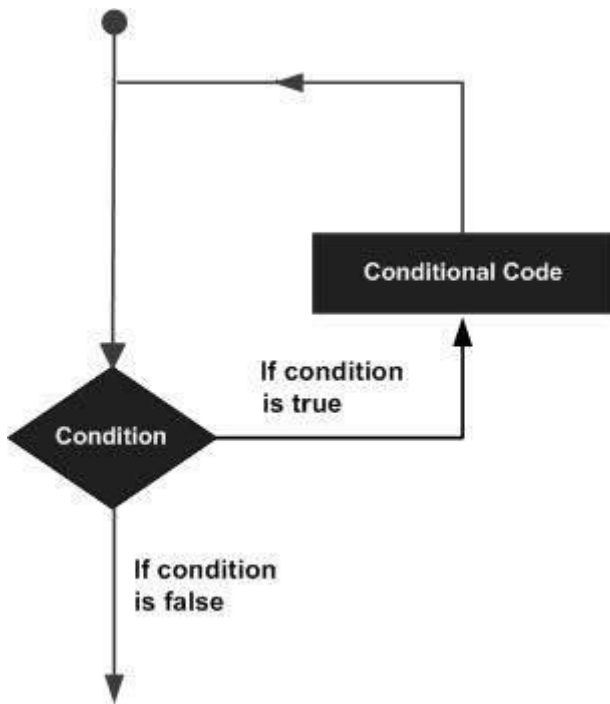
```
Value of expression is 100  
Good bye!
```

Python - Bucles

En general, las instrucciones se ejecutan secuencialmente: la primera instrucción de una función se ejecuta primero, seguida de la segunda, y así sucesivamente. Puede haber una situación en la que necesite ejecutar un bloque de código varias veces.

Los lenguajes de programación proporcionan diversas estructuras de control que permiten rutas de ejecución más complicadas.

Una declaración de bucle nos permite ejecutar una declaración o grupo de declaraciones varias veces. El siguiente diagrama ilustra una declaración de bucle:



El lenguaje de programación Python proporciona los siguientes tipos de bucles para manejar los requisitos de bucle.

No Señor.	Tipo de bucle y descripción
1	<u>mientras bucle</u> Repite una declaración o grupo de declaraciones mientras una condición dada es VERDADERA. Prueba la condición antes de ejecutar el cuerpo del bucle.
2	<u>en bucle</u> Ejecuta una secuencia de declaraciones varias veces y abrevia el código que administra la variable de bucle.
3	<u>bucles anidados</u> Puede usar uno o más bucles dentro de otro mientras, para o hacer ... while loop.

Declaraciones de control de bucle

Las instrucciones de control de bucle cambian la ejecución de su secuencia normal. Cuando la ejecución deja un ámbito, todos los objetos automáticos que se crearon en ese ámbito se destruyen.

Python admite las siguientes declaraciones de control. Haga clic en los siguientes enlaces para verificar sus detalles.

Veamos brevemente las declaraciones de control de bucle

No Señor.	Declaración de control y descripción
1	<u>declaración de ruptura</u> Termina la instrucción de bucle y transfiere la ejecución a la instrucción que sigue inmediatamente al bucle.
2	<u>continuar declaración</u> Hace que el bucle omita el resto de su cuerpo e inmediatamente vuelva a probar su condición antes de reiterar.
3	<u>declaración de aprobación</u> La sentencia pass en Python se usa cuando se requiere una sentencia sintácticamente pero no desea que se ejecute ningún comando o código.

Python - Números

Los tipos de datos numéricos almacenan valores numéricos. Son tipos de datos inmutables, lo que significa que cambiar el valor de un tipo de datos numérico da como resultado un objeto recientemente asignado.

Los objetos numéricos se crean cuando les asigna un valor. Por ejemplo

```
var1 = 1
var2 = 10
```

También puede eliminar la referencia a un objeto numérico utilizando la instrucción **del**. La sintaxis de la declaración del es -

```
del var1[,var2[,var3[....,varN]]]
```

Puede eliminar un solo objeto o varios objetos utilizando la instrucción **del**. Por ejemplo

```
del var
del var_a, var_b
```

Python admite cuatro tipos numéricos diferentes:

- **int (enteros con signo)** : a menudo se los llama enteros o enteros, son números enteros positivos o negativos sin punto decimal.
- **long (enteros largos)** : también llamados largos, son enteros de tamaño ilimitado, escritos como enteros y seguidos de una L mayúscula o minúscula.
- **flotante (valores reales de coma flotante)** : también llamados flotantes, representan números reales y se escriben con un punto decimal que divide el número entero y las partes fraccionarias. Los flotadores también pueden estar en notación científica, con E o e indicando la potencia de 10 ($2.5e2 = 2.5 \times 10^2 = 250$).
- **complejo (números complejos)** : son de la forma $a + bJ$, donde a y b son flotantes y J (o j) representa la raíz cuadrada de -1 (que es un número

imaginario). La parte real del número es a, y la parte imaginaria es b. Los números complejos no se usan mucho en la programación de Python.

Ejemplos

Aquí hay algunos ejemplos de números.

En t	largo	flotador	complejo
10	51924361L	0.0	3.14j
100	-0x19323L	15,20	45.j
-786	0122L	-21,9	9.322e-36j
080	0xDEFABCECBDAECBFBAEL	32,3 + e18	.876j
-0490	535633629843L	-90.	-.6545 + 0J
-0x260	-052318172735L	-32.54e100	3e + 26J
0x69	-4721885298529L	70.2-E12	4.53e-7j

- Python le permite usar una L minúscula con largo, pero se recomienda que use solo una L mayúscula para evitar confusiones con el número 1. Python muestra enteros largos con una L mayúscula.
- Un número complejo consiste en un par ordenado de números de coma flotante reales denotados por a + bj, donde a es la parte real y b es la parte imaginaria del número complejo.

Conversión de tipo de número

Python convierte números internamente en una expresión que contiene tipos mixtos a un tipo común para evaluación. Pero a veces, debe forzar un número explícitamente de un tipo a otro para satisfacer los requisitos de un operador o parámetro de función.

- Escriba **int (x)** para convertir x en un entero simple.
- Escriba **long (x)** para convertir x en un entero largo.
- Escriba **float (x)** para convertir x en un número de coma flotante.
- Escriba **complex (x)** para convertir x en un número complejo con parte real xy parte cero imaginaria.

- Escriba **complex (x, y)** para convertir xey en un número complejo con parte real xy parte imaginaria y. x e y son expresiones numéricas

Funciones matematicas

Python incluye las siguientes funciones que realizan cálculos matemáticos.

No Señor.	Función y devoluciones (descripción)
1	<u>abs (x)</u> El valor absoluto de x: la distancia (positiva) entre x y cero.
2	<u>techo (x)</u> El techo de x: el entero más pequeño no menos que x
3	<u>cmp (x, y)</u> -1 si $x < y$, 0 si $x == y$, o 1 si $x > y$
4 4	<u>exp (x)</u> El exponencial de x: e^x
5 5	<u>fabs (x)</u> El valor absoluto de x.
6 6	<u>piso (x)</u> El piso de x: el entero más grande no mayor que x
7 7	<u>log (x)</u> El logaritmo natural de x, para $x > 0$
8	<u>log10 (x)</u> El logaritmo de base 10 de x para $x > 0$.
9 9	<u>max (x1, x2, ...)</u> El mayor de sus argumentos: el valor más cercano al infinito positivo
10	<u>min (x1, x2, ...)</u> El más pequeño de sus argumentos: el valor más cercano al infinito negativo
11	<u>modf (x)</u> Las partes fraccionarias y enteras de x en una tupla de dos elementos. Ambas

	partes tienen el mismo signo que x. La parte entera se devuelve como flotante.
12	<u>pow (x, y)</u> El valor de $x^{**}y$.
13	<u>redondo (x [, n])</u> x redondeado a n dígitos desde el punto decimal. Python se redondea desde cero como un desempate: la ronda (0.5) es 1.0 y la ronda (-0.5) es -1.0.
14	<u>sqrt (x)</u> La raíz cuadrada de x para $x > 0$

Funciones de números aleatorios

Los números aleatorios se utilizan para juegos, simulaciones, pruebas, aplicaciones de seguridad y privacidad. Python incluye las siguientes funciones que se usan comúnmente.

No Señor.	Función descriptiva
1	<u>elección (seq)</u> Un elemento aleatorio de una lista, tupla o cadena.
2	<u>randrange ([inicio,] parada [, paso])</u> Un elemento seleccionado al azar del rango (inicio, parada, paso)
3	<u>aleatorio()</u> Un flotador aleatorio r, tal que 0 es menor o igual que r y r es menor que 1
4 4	<u>semilla ([x])</u> Establece el valor de inicio entero usado para generar números aleatorios. Llame a esta función antes de llamar a cualquier otra función de módulo aleatorio. Devuelve ninguno.
5 5	<u>barajar (lst)</u> Aleatoriza los elementos de una lista en su lugar. Devuelve ninguno.
6 6	<u>uniforme (x, y)</u> Un flotador aleatorio r, tal que x es menor o igual que r y r es menor que y

Funciones trigonométricas

Python incluye las siguientes funciones que realizan cálculos trigonométricos.

No Señor.	Función descriptiva
1	<u>acos (x)</u> Devuelve el arco coseno de x, en radianes.
2	<u>asin (x)</u> Devuelve el arco seno de x, en radianes.
3	<u>atan (x)</u> Devuelve el arco tangente de x, en radianes.
4 4	<u>atan2 (y, x)</u> Devuelve atan (y / x), en radianes.
5 5	<u>cos (x)</u> Devuelve el coseno de x radianes.
6 6	<u>hipot (x, y)</u> Devuelve la norma euclidiana, $\sqrt{x^2 + y^2}$.
7 7	<u>sin (x)</u> Devuelve el seno de x radianes.
8	<u>bronceado (x)</u> Devuelve la tangente de x radianes.
9 9	<u>grados (x)</u> Convierte el ángulo x de radianes a grados.
10	<u>radianes (x)</u> Convierte el ángulo x de grados a radianes.

Constantes Matemáticas

El módulo también define dos constantes matemáticas:

No Señor.	Constantes y Descripción
1	Pi La constante matemática pi.
2	mi La constante matemática e.

Python - Cadenas

Las cadenas se encuentran entre los tipos más populares en Python. Podemos crearlos simplemente encerrando caracteres entre comillas. Python trata las comillas simples igual que las comillas dobles. Crear cadenas es tan simple como asignar un valor a una variable. Por ejemplo

```
var1 = 'Hello World!'
var2 = "Python Programming"
```

Acceso a valores en cadenas

Python no admite un tipo de carácter; estos se tratan como cadenas de longitud uno, por lo que también se consideran una subcadena.

Para acceder a las subcadenas, use corchetes para cortar junto con el índice o los índices para obtener su subcadena. Por ejemplo

```
#!/usr/bin/python

var1 = 'Hello World!'
var2 = "Python Programming"

print "var1[0]: ", var1[0]
print "var2[1:5]: ", var2[1:5]
```

Cuando se ejecuta el código anterior, produce el siguiente resultado:

```
var1[0]:  H
var2[1:5]:  ytho
```

Actualización de cadenas

Puede "actualizar" una cadena existente (re) asignando una variable a otra cadena. El nuevo valor puede estar relacionado con su valor anterior o con una cadena completamente diferente. Por ejemplo

```
#!/usr/bin/python  
  
var1 = 'Hello World!'  
print "Updated String :- ", var1[:6] + 'Python'
```

Cuando se ejecuta el código anterior, produce el siguiente resultado:

```
Updated String :- Hello Python
```

Personajes de escape

La siguiente tabla es una lista de caracteres de escape o no imprimibles que se pueden representar con notación de barra invertida.

Un personaje de escape se interpreta; en cadenas de comillas simples y comillas dobles.

Notación de barra invertida	Carácter hexadecimal	Descripción
\un	0x07	Campana o alerta
\si	0x08	Retroceso
\cx		Control-x
\Cx		Control-x
\mi	0x1b	Escapar
\F	0x0c	Formfeed
\M-\Cx		Meta-Control-x
\norte	0x0a	Nueva línea

\nnn		Notación octal, donde n está en el rango 0.7
\r	0x0d	Retorno de carro
\s	0x20	Espacio
\t	0x09	Lengüeta
\v	0x0b	Pestaña vertical
\x		Personaje x
\xnn		Notación hexadecimal, donde n está en el rango 0.9, af o AF

Operadores especiales de cadenas

Suponga que la variable de cadena **a** contiene 'Hola' y la variable **b** contiene 'Python', luego -

Operador	Descripción	Ejemplo
+	Concatenación: agrega valores a ambos lados del operador	a + b dará HelloPython
**	Repetición: crea nuevas cadenas, concatenando múltiples copias de la misma cadena	a * 2 dará - HolaHola
[]	Slice - Da el personaje del índice dado	a [1] dará e
[:]	Range Slice - Da los caracteres del rango dado	a [1: 4] dará ell
en	Membresía: devuelve verdadero si existe un carácter en la cadena dada	H en a dará 1

no en	Membresía: devuelve verdadero si un carácter no existe en la cadena dada	M no en una voluntad dará 1
r / R	Cadena sin formato: suprime el significado real de los caracteres de escape. La sintaxis para las cadenas sin formato es exactamente la misma que para las cadenas normales, con la excepción del operador de cadena sin formato, la letra "r", que precede a las comillas. La "r" puede ser minúscula (r) o mayúscula (R) y debe colocarse inmediatamente antes de la primera comilla.	imprime r '\n' imprime \ny imprime R '\n' n'impressiones \n
%	Formato: realiza el formateo de cadenas	Ver en la siguiente sección

Operador de formateo de cadenas

Una de las mejores características de Python es el operador de formato de cadena%. Este operador es exclusivo de las cadenas y compensa el paquete de funciones de la familia printf () de C. El siguiente es un ejemplo simple:

```
#!/usr/bin/python
print "My name is %s and weight is %d kg!" % ('Zara', 21)
```

Cuando se ejecuta el código anterior, produce el siguiente resultado:

```
My name is Zara and weight is 21 kg!
```

Aquí está la lista del conjunto completo de símbolos que se pueden usar junto con% -

Símbolo de formato	Conversión
%C	personaje
%s	conversión de cadena a través de str () antes de formatear
%yo	entero decimal con signo
%re	entero decimal con signo

% u	entero decimal sin signo
% o	entero octal
%X	entero hexadecimal (letras minúsculas)
%X	entero hexadecimal (MAYÚSCULAS)
%mi	notación exponencial (con minúscula 'e')
%MI	notación exponencial (con MAYÚSCULAS 'E')
%F	número real de coma flotante
%sol	el más corto de% f y% e
%SOL	el más corto de% f y% E

Otros símbolos y funcionalidades compatibles se enumeran en la siguiente tabla:

Símbolo	Funcionalidad
* *	argumento especifica ancho o precisión
-	justificación izquierda
+	muestra el letrero
<sp>	deje un espacio en blanco antes de un número positivo
# #	agregue el cero inicial octal ('0') o el hexadecimal inicial '0x' o '0X', dependiendo de si se usaron 'x' o 'X'.
0 0	pad desde la izquierda con ceros (en lugar de espacios)

%	'%%' te deja con un solo literal '%'
(var)	variable de mapeo (argumentos del diccionario)
Minnesota	m es el ancho total mínimo y n es el número de dígitos que se mostrarán después del punto decimal (si corresponde).

Cotizaciones triples

Las citas triples de Python vienen al rescate al permitir que las cadenas abarquen varias líneas, incluidas NEWLINE, TAB y cualquier otro carácter especial.

La sintaxis para las comillas triples consta de tres comillas **simples** o **dobles** consecutivas .

```
#!/usr/bin/python

para_str = """this is a long string that is made up of
several lines and non-printable characters such as
TAB ( \t ) and they will show up that way when displayed.
NEWLINEs within the string, whether explicitly given like
this within the brackets [ \n ], or just a NEWLINE within
the variable assignment will also show up.
"""
print para_str
```

Cuando se ejecuta el código anterior, produce el siguiente resultado. Observe cómo cada carácter especial se ha convertido a su forma impresa, hasta el último NEWLINE al final de la cadena entre "arriba". y cierre de comillas triples. También tenga en cuenta que NEWLINE se produce con un retorno de carro explícito al final de una línea o con su código de escape (\n) -

```
this is a long string that is made up of
several lines and non-printable characters such as
TAB (   ) and they will show up that way when displayed.
NEWLINEs within the string, whether explicitly given like
this within the brackets [
 ], or just a NEWLINE within
the variable assignment will also show up.
```

Las cadenas sin formato no tratan la barra invertida como un carácter especial en absoluto. Cada carácter que pones en una cadena sin procesar se mantiene como lo escribiste:

```
#!/usr/bin/python
```

```
print 'C:\\nowhere'
```

Cuando se ejecuta el código anterior, produce el siguiente resultado:

```
C:\\nowhere
```

Ahora hagamos uso de una cadena cruda. **Pondríamos** expresión en **r'expression'** de la siguiente manera:

```
#!/usr/bin/python  
print r'C:\\nowhere'
```

Cuando se ejecuta el código anterior, produce el siguiente resultado:

```
C:\\nowhere
```

Cadena Unicode

Las cadenas normales en Python se almacenan internamente como ASCII de 8 bits, mientras que las cadenas Unicode se almacenan como Unicode de 16 bits. Esto permite un conjunto más variado de caracteres, incluidos caracteres especiales de la mayoría de los idiomas del mundo. Restringiré mi tratamiento de cadenas Unicode a lo siguiente:

```
#!/usr/bin/python  
print u'Hello, world!'
```

Cuando se ejecuta el código anterior, produce el siguiente resultado:

```
Hello, world!
```

Como puede ver, las cadenas Unicode usan el prefijo u, así como las cadenas sin procesar usan el prefijo r.

Métodos de cadena incorporados

Python incluye los siguientes métodos integrados para manipular cadenas:

No Señor.	Métodos con descripción
1	<u>capitalizar()</u> Capitaliza la primera letra de cadena
2	<u>centro (ancho, relleno)</u> Devuelve una cadena de espacio con la cadena original centrada en un total de columnas de ancho.

3	<p><u>cuenta (str, beg = 0, end = len (string))</u></p> <p>Cuenta cuántas veces se produce una cadena en una cadena o en una subcadena de cadena si se da inicio al inicio del índice y finalización del índice.</p>
4 4	<p><u>decodificación (codificación = 'UTF-8', errores = 'estricto')</u></p> <p>Decodifica la cadena usando el códec registrado para la codificación. la codificación predeterminada es la codificación de cadena predeterminada.</p>
5 5	<p><u>codificar (codificación = 'UTF-8', errores = 'estricto')</u></p> <p>Devuelve la versión de cadena codificada de cadena; en caso de error, el valor predeterminado es generar un ValueError a menos que los errores se den con 'ignorar' o 'reemplazar'.</p>
6 6	<p><u>endswith (sufijo, beg = 0, end = len (cadena))</u></p> <p>Determina si la cadena o una subcadena de cadena (si se da inicio al inicio del índice y finalización del índice final) termina con sufijo; devuelve verdadero si es así y falso de lo contrario.</p>
7 7	<p><u>expandtabs (tabsize = 8)</u></p> <p>Expande pestañas en cadena a múltiples espacios; el valor predeterminado es 8 espacios por pestaña si no se proporciona el tamaño de la pestaña.</p>
8	<p><u>find (str, beg = 0 end = len (cadena))</u></p> <p>Determine si la cadena se produce en una cadena o en una subcadena de la cadena si el índice inicial comienza y finaliza el índice final devuelve índice si se encuentra y -1 de lo contrario.</p>
9 9	<p><u>index (str, beg = 0, end = len (string))</u></p> <p>Igual que find (), pero genera una excepción si no se encuentra str.</p>
10	<p><u>isalnum ()</u></p> <p>Devuelve verdadero si la cadena tiene al menos 1 carácter y, de lo contrario, todos los caracteres son alfanuméricos y falsos.</p>
11	<p><u>isalpha ()</u></p> <p>Devuelve verdadero si la cadena tiene al menos 1 carácter y, de lo contrario, todos los caracteres son alfabéticos y falsos.</p>
12	<p><u>isdigit ()</u></p> <p>Devuelve verdadero si la cadena contiene solo dígitos y falso en caso contrario.</p>
13	<p><u>es bajo()</u></p> <p>Devuelve verdadero si la cadena tiene al menos 1 carácter en mayúscula y, de lo contrario, todos los caracteres en mayúscula están en minúscula y falso.</p>

14	<u>isnumeric ()</u> Devuelve verdadero si una cadena unicode contiene solo caracteres numéricos y falso en caso contrario.
15	<u>isspace ()</u> Devuelve verdadero si la cadena contiene solo caracteres de espacio en blanco y falso en caso contrario.
dieciséis	<u>istitle ()</u> Devuelve verdadero si la cadena está correctamente "encasillada" y falso en caso contrario
17	<u>isupper ()</u> Devuelve verdadero si la cadena tiene al menos un carácter en mayúscula y, de lo contrario, todos los caracteres en mayúscula están en mayúscula y falso.
18	<u>unirse (seq)</u> Fusiona (concatena) las representaciones de cadena de elementos en secuencia seq en una cadena, con una cadena de separación.
19	<u>len (cadena)</u> Devuelve la longitud de la cadena.
20	<u>ljust (ancho [, fillchar])</u> Devuelve una cadena de espacio con la cadena original justificada a la izquierda a un total de columnas de ancho.
21	<u>inferior()</u> Convierte todas las letras mayúsculas en cadena a minúsculas.
22	<u>lstrip ()</u> Elimina todos los espacios en blanco iniciales en la cadena.
23	<u>maketrans ()</u> Devuelve una tabla de traducción para usar en la función de traducción.
24	<u>max (str)</u> Devuelve el carácter alfabético máximo de la cadena str.
25	<u>min (str)</u> Devuelve el carácter alfabético mínimo de la cadena str.
26	<u>reemplazar (antiguo, nuevo [, máx.])</u>

	Reemplaza todas las ocurrencias de edad en cadena con nuevas o como máximo, si se da el máximo.
27	<u>rfind (str, beg = 0, end = len (cadena))</u> Igual que find (), pero busca hacia atrás en cadena.
28	<u>rindex (str, beg = 0, end = len (string))</u> Igual que index (), pero busca hacia atrás en la cadena.
29	<u>rjust (ancho, [, fillchar])</u> Devuelve una cadena de espacio con la cadena original justificada a la derecha a un total de columnas de ancho.
30	<u>rstrip ()</u> Elimina todo el espacio en blanco al final de la cadena.
31	<u>split (str = "", num = string.count (str))</u> Divide la cadena de acuerdo con la cadena del delimitador (espacio si no se proporciona) y devuelve la lista de subcadenas; dividir en la mayoría de las subcadenas numéricas si se da.
32	<u>splitlines (num = string.count ('\ n'))</u> Divide la cadena en todas las NEWLINE (o num.) Y devuelve una lista de cada línea con NEWLINE eliminadas.
33	<u>comienza con (str, beg = 0, end = len (string))</u> Determina si la cadena o una subcadena de cadena (si se dan inicio del índice inicial y final del índice final) comienza con la cadena de subcadena; devuelve verdadero si es así y falso de lo contrario.
34	<u>tira ([caracteres])</u> Realiza tanto lstrip () como rstrip () en cadena.
35	<u>swapcase ()</u> Invierte mayúsculas y minúsculas para todas las letras en cadena.
36	<u>título()</u> Devuelve la versión "stringcased" de la cadena, es decir, todas las palabras comienzan con mayúsculas y el resto son minúsculas.
37	<u>translate (table, deletchars = "")</u> Traduce la cadena de acuerdo con la tabla de traducción str (256 caracteres), eliminando aquellos en la cadena del.

38	<u>Superior()</u> Convierte letras minúsculas en cadena a mayúsculas.
39	<u>zfill (ancho)</u> Devuelve la cadena original rellena con ceros a la izquierda a un total de caracteres de ancho; destinado a números, zfill () retiene cualquier signo dado (menos un cero).
40	<u>isdecimal ()</u> Devuelve verdadero si una cadena unicode contiene solo caracteres decimales y falso en caso contrario.

Python - Listas

La estructura de datos más básica en Python es la **secuencia**. A cada elemento de una secuencia se le asigna un número: su posición o índice. El primer índice es cero, el segundo índice es uno, y así sucesivamente.

Python tiene seis tipos de secuencias incorporadas, pero las más comunes son listas y tuplas, que veremos en este tutorial.

Hay ciertas cosas que puede hacer con todos los tipos de secuencia. Estas operaciones incluyen indexación, segmentación, adición, multiplicación y verificación de membresía. Además, Python tiene funciones integradas para encontrar la longitud de una secuencia y para encontrar sus elementos más grandes y más pequeños.

Listas de Python

La lista es un tipo de datos más versátil disponible en Python que se puede escribir como una lista de valores (elementos) separados por comas entre corchetes. Lo importante de una lista es que los elementos de una lista no necesitan ser del mismo tipo.

Crear una lista es tan simple como poner diferentes valores separados por comas entre corchetes. Por ejemplo

```
list1 = ['physics', 'chemistry', 1997, 2000];
list2 = [1, 2, 3, 4, 5 ];
list3 = ["a", "b", "c", "d"]
```

Al igual que los índices de cadena, los índices de lista comienzan en 0, y las listas se pueden dividir, concatenar, etc.

Acceso a valores en listas

Para acceder a los valores en las listas, use corchetes para segmentar junto con el índice o los índices para obtener el valor disponible en ese índice. Por ejemplo

```
#!/usr/bin/python

list1 = ['physics', 'chemistry', 1997, 2000];
list2 = [1, 2, 3, 4, 5, 6, 7 ];
print "list1[0]: ", list1[0]
print "list2[1:5]: ", list2[1:5]
```

Cuando se ejecuta el código anterior, produce el siguiente resultado:

```
list1[0]: physics
list2[1:5]: [2, 3, 4, 5]
```

Actualización de listas

Puede actualizar elementos simples o múltiples de listas dando el segmento en el lado izquierdo del operador de asignación, y puede agregar elementos a una lista con el método `append ()`. Por ejemplo

```
#!/usr/bin/python

list = ['physics', 'chemistry', 1997, 2000];
print "Value available at index 2 : "
print list[2]
list[2] = 2001;
print "New value available at index 2 : "
print list[2]
```

Nota : el método `append ()` se trata en la sección siguiente.

Cuando se ejecuta el código anterior, produce el siguiente resultado:

```
Value available at index 2 :
1997
New value available at index 2 :
2001
```

Eliminar elementos de la lista

Para eliminar un elemento de la lista, puede usar la instrucción `del` si sabe exactamente qué elemento (s) está eliminando o el método `remove ()` si no lo sabe. Por ejemplo

```
#!/usr/bin/python

list1 = ['physics', 'chemistry', 1997, 2000];
print list1
del list1[2];
print "After deleting value at index 2 : "
print list1
```

Cuando se ejecuta el código anterior, produce el siguiente resultado:

```
['physics', 'chemistry', 1997, 2000]
After deleting value at index 2 :
['physics', 'chemistry', 2000]
```

Nota : el método remove () se trata en la sección siguiente.

Operaciones básicas de lista

Las listas responden a los operadores + y * de forma muy similar a las cadenas; aquí también significan concatenación y repetición, excepto que el resultado es una nueva lista, no una cadena.

De hecho, las listas responden a todas las operaciones de secuencia general que utilizamos en cadenas en el capítulo anterior.

Expresión de Python	Resultados	Descripción
len ([1, 2, 3])	3	Longitud
[1, 2, 3] + [4, 5, 6]	[1, 2, 3, 4, 5, 6]	Concatenación
['¡Hola!'] * 4	['¡Hola!', '¡Hola!', '¡Hola!', '¡Hola!']	Repetición
3 en [1, 2, 3]	Cierto	Afiliación
para x en [1, 2, 3]: imprimir x,	1 2 3	Iteración

Indexación, segmentación y matrices

Como las listas son secuencias, la indexación y el corte funcionan de la misma manera para las listas que para las cadenas.

Suponiendo la siguiente entrada:

```
L = ['spam', 'Spam', 'SPAM!']
```

Expresión de Python	Resultados	Descripción
L [2]	¡CORREO NO DESEADO!	Las compensaciones comienzan en cero
L [-2]	Correo no deseado	Negativo: cuenta desde la derecha

L [1:]	['Spam', 'SPAM!']	Cortar secciones de recuperaciones
--------	-------------------	------------------------------------

Lista incorporada de funciones y métodos

Python incluye las siguientes funciones de lista:

No Señor.	Función con descripción
1	<u>cmp (lista1, lista2)</u> Compara elementos de ambas listas.
2	<u>len (lista)</u> Da la longitud total de la lista.
3	<u>max (lista)</u> Devuelve el artículo de la lista con el valor máximo.
4 4	<u>min (lista)</u> Devuelve el artículo de la lista con el valor mínimo.
5 5	<u>lista (seq)</u> Convierte una tupla en lista.

Python incluye los siguientes métodos de lista

No Señor.	Métodos con descripción
1	<u>list.append (obj)</u> Agrega objeto obj a la lista
2	<u>list.count (obj)</u> Devuelve el recuento de cuántas veces se produce obj en la lista
3	<u>list.extend (seq)</u> Agrega los contenidos de seq a la lista
4 4	<u>list.index (obj)</u>

	Devuelve el índice más bajo de la lista en el que aparece obj
5 5	<u>list.insert (index, obj)</u> Inserta el objeto obj en la lista en el índice de desplazamiento
6 6	<u>list.pop (obj = lista [-1])</u> Elimina y devuelve el último objeto u obj de la lista
7 7	<u>list.remove (obj)</u> Elimina el objeto obj de la lista
8	<u>list.reverse ()</u> Invierte objetos de la lista en su lugar
9 9	<u>list.sort ([func])</u> Ordena los objetos de la lista, usa la función de comparación si se proporciona

Python - Tuplas

Una tupla es una secuencia de objetos Python inmutables. Las tuplas son secuencias, al igual que las listas. Las diferencias entre las tuplas y las listas son que las tuplas no se pueden cambiar a diferencia de las listas y las tuplas usan paréntesis, mientras que las listas usan corchetes.

Crear una tupla es tan simple como poner diferentes valores separados por comas. Opcionalmente, también puede poner estos valores separados por comas entre paréntesis. Por ejemplo

```
tup1 = ('physics', 'chemistry', 1997, 2000);
tup2 = (1, 2, 3, 4, 5 );
tup3 = "a", "b", "c", "d";
```

La tupla vacía se escribe como dos paréntesis que no contienen nada:

```
tup1 = ();
```

Para escribir una tupla que contenga un único valor, debe incluir una coma, aunque solo haya un valor:

```
tup1 = (50,);
```

Al igual que los índices de cadena, los índices de tupla comienzan en 0, y se pueden dividir, concatenar, etc.

Acceso a valores en tuplas

Para acceder a los valores en tupla, use corchetes para segmentar junto con el índice o los índices para obtener el valor disponible en ese índice. Por ejemplo

```
#!/usr/bin/python

tup1 = ('physics', 'chemistry', 1997, 2000);
tup2 = (1, 2, 3, 4, 5, 6, 7 );
print "tup1[0]: ", tup1[0];
print "tup2[1:5]: ", tup2[1:5];
```

Cuando se ejecuta el código anterior, produce el siguiente resultado:

```
tup1[0]: physics
tup2[1:5]: [2, 3, 4, 5]
```

Actualización de tuplas

Las tuplas son inmutables, lo que significa que no puede actualizar o cambiar los valores de los elementos de tupla. Puede tomar porciones de tuplas existentes para crear nuevas tuplas como lo demuestra el siguiente ejemplo:

```
#!/usr/bin/python

tup1 = (12, 34.56);
tup2 = ('abc', 'xyz');

# Following action is not valid for tuples
# tup1[0] = 100;

# So let's create a new tuple as follows
tup3 = tup1 + tup2;
print tup3;
```

Cuando se ejecuta el código anterior, produce el siguiente resultado:

```
(12, 34.56, 'abc', 'xyz')
```

Eliminar elementos de tupla

No es posible eliminar elementos de tupla individuales. Por supuesto, no hay nada de malo en juntar otra tupla con los elementos no deseados descartados.

Para eliminar explícitamente una tupla completa, solo use la instrucción **del** . Por ejemplo

```
#!/usr/bin/python

tup = ('physics', 'chemistry', 1997, 2000);
print tup;
del tup;
print "After deleting tup : ";
print tup;
```

Esto produce el siguiente resultado. Tenga en cuenta una excepción planteada, esto se debe a que después de **tup** tuple ya no existe:

```
('physics', 'chemistry', 1997, 2000)
After deleting tup :
Traceback (most recent call last):
  File "test.py", line 9, in <module>
    print tup;
NameError: name 'tup' is not defined
```

Operaciones básicas de tuplas

Las tuplas responden a los operadores + y * de forma muy similar a las cadenas; aquí también significan concatenación y repetición, excepto que el resultado es una nueva tupla, no una cadena.

De hecho, las tuplas responden a todas las operaciones de secuencia general que utilizamos en cadenas en el capítulo anterior:

Expresión de Python	Resultados	Descripción
len ((1, 2, 3))	3	Longitud
(1, 2, 3) + (4, 5, 6)	(1, 2, 3, 4, 5, 6)	Concatenación
('¡Hola!') * 4	('¡Hola!', '¡Hola!', '¡Hola!', '¡Hola!')	Repetición
3 pulgadas (1, 2, 3)	Cierto	Afiliación
para x in (1, 2, 3): imprimir x,	1 2 3	Iteración

Indexación, segmentación y matrices

Como las tuplas son secuencias, la indexación y el corte funcionan de la misma manera para las tuplas que para las cadenas. Suponiendo la siguiente entrada:

```
L = ('spam', 'Spam', 'SPAM!')
```

Expresión de Python	Resultados	Descripción
---------------------	------------	-------------

L [2]	'¡CORREO NO DESEADO!'	Las compensaciones comienzan en cero
L [-2]	'Correo no deseado'	Negativo: cuenta desde la derecha
L [1:]	['Spam', 'SPAM!']	Cortar secciones de recuperaciones

Sin delimitadores envolventes

Cualquier conjunto de objetos múltiples, separados por comas, escritos sin símbolos de identificación, es decir, corchetes para listas, paréntesis para tuplas, etc., por defecto a tuplas, como se indica en estos ejemplos cortos:

```
#!/usr/bin/python
print 'abc', -4.24e93, 18+6.6j, 'xyz';
x, y = 1, 2;
print "Value of x , y : ", x,y;
```

Cuando se ejecuta el código anterior, produce el siguiente resultado:

```
abc -4.24e+93 (18+6.6j) xyz
Value of x , y : 1 2
```

Funciones de tupla incorporadas

Python incluye las siguientes funciones de tupla:

No Señor.	Función con descripción
1	<u>cmp (tupla1, tupla2)</u> Compara elementos de ambas tuplas.
2	<u>len (tupla)</u> Da la longitud total de la tupla.
3	<u>max (tupla)</u> Devuelve el artículo de la tupla con el valor máximo.
4 4	<u>min (tupla)</u> Devuelve el artículo de la tupla con el valor mínimo.

5 5

tupla (seq)

Convierte una lista en tupla.

Python - Diccionario

Cada tecla está separada de su valor por dos puntos (:), los elementos están separados por comas y todo está encerrado entre llaves. Un diccionario vacío sin ningún elemento se escribe con solo dos llaves, así: {}.

Las claves son únicas dentro de un diccionario, mientras que los valores pueden no serlo. Los valores de un diccionario pueden ser de cualquier tipo, pero las claves deben ser de un tipo de datos inmutable, como cadenas, números o tuplas.

Acceso a valores en el diccionario

Para acceder a los elementos del diccionario, puede usar los corchetes familiares junto con la clave para obtener su valor. El siguiente es un ejemplo simple:

```
#!/usr/bin/python

dict = {'Name': 'Zara', 'Age': 7, 'Class': 'First'}
print "dict['Name']: ", dict['Name']
print "dict['Age']: ", dict['Age']
```

Cuando se ejecuta el código anterior, produce el siguiente resultado:

```
dict['Name']: Zara
dict['Age']: 7
```

Si intentamos acceder a un elemento de datos con una clave, que no es parte del diccionario, obtenemos el siguiente error:

```
#!/usr/bin/python

dict = {'Name': 'Zara', 'Age': 7, 'Class': 'First'}
print "dict['Alice']: ", dict['Alice']
```

Cuando se ejecuta el código anterior, produce el siguiente resultado:

```
dict['Alice']:
Traceback (most recent call last):
  File "test.py", line 4, in <module>
    print "dict['Alice']: ", dict['Alice'];
KeyError: 'Alice'
```

Diccionario de actualización

Puede actualizar un diccionario agregando una nueva entrada o un par clave-valor, modificando una entrada existente o eliminando una entrada existente como se muestra a continuación en el ejemplo simple:

```
#!/usr/bin/python

dict = {'Name': 'Zara', 'Age': 7, 'Class': 'First'}
dict['Age'] = 8; # update existing entry
dict['School'] = "DPS School"; # Add new entry

print "dict['Age']: ", dict['Age']
print "dict['School']: ", dict['School']
```

Cuando se ejecuta el código anterior, produce el siguiente resultado:

```
dict['Age']: 8
dict['School']: DPS School
```

Eliminar elementos del diccionario

Puede eliminar elementos individuales del diccionario o borrar todo el contenido de un diccionario. También puede eliminar todo el diccionario en una sola operación.

Para eliminar explícitamente un diccionario completo, solo use la instrucción **del** . El siguiente es un ejemplo simple:

```
#!/usr/bin/python

dict = {'Name': 'Zara', 'Age': 7, 'Class': 'First'}
del dict['Name']; # remove entry with key 'Name'
dict.clear();    # remove all entries in dict
del dict ;      # delete entire dictionary

print "dict['Age']: ", dict['Age']
print "dict['School']: ", dict['School']
```

Esto produce el siguiente resultado. Tenga en cuenta que se genera una excepción porque después **del** diccionario **dict** ya no existe:

```
dict['Age']:
Traceback (most recent call last):
  File "test.py", line 8, in <module>
    print "dict['Age']: ", dict['Age'];
TypeError: 'type' object is unsubscriptable
```

Nota : el método `del ()` se analiza en la sección siguiente.

Propiedades de las claves del diccionario

Los valores del diccionario no tienen restricciones. Pueden ser cualquier objeto arbitrario de Python, ya sea objetos estándar u objetos definidos por el usuario. Sin embargo, lo mismo no es cierto para las claves.

Hay dos puntos importantes para recordar acerca de las teclas del diccionario:

(a) Más de una entrada por clave no permitida. Lo que significa que no se permite duplicar la clave. Cuando se encuentran claves duplicadas durante la asignación, gana la última asignación. Por ejemplo

```
#!/usr/bin/python
dict = {'Name': 'Zara', 'Age': 7, 'Name': 'Manni'}
print "dict['Name']: ", dict['Name']
```

Cuando se ejecuta el código anterior, produce el siguiente resultado:

```
dict['Name']: Manni
```

(b) Las llaves deben ser inmutables. Lo que significa que puede usar cadenas, números o tuplas como claves de diccionario, pero algo como ['clave'] no está permitido. El siguiente es un ejemplo simple:

```
#!/usr/bin/python
dict = {'Name': 'Zara', 'Age': 7}
print "dict['Name']: ", dict['Name']
```

Cuando se ejecuta el código anterior, produce el siguiente resultado:

```
Traceback (most recent call last):
  File "test.py", line 3, in <module>
    dict = {'Name': 'Zara', 'Age': 7};
TypeError: unhashable type: 'list'
```

Funciones y métodos de diccionario incorporados

Python incluye las siguientes funciones de diccionario:

No Señor.	Función con descripción
1	<u>cmp (dict1, dict2)</u> Compara elementos de ambos dict.
2	<u>len (dict)</u> Da la longitud total del diccionario. Esto sería igual al número de elementos en el

	diccionario.
3	<u>str (dict)</u> Produce una representación de cadena imprimible de un diccionario
4 4	<u>tipo (variable)</u> Devuelve el tipo de la variable pasada. Si la variable pasada es diccionario, entonces devolvería un tipo de diccionario.

Python incluye los siguientes métodos de diccionario:

No Señor.	Métodos con descripción
1	<u>dict.clear ()</u> Elimina todos los elementos del diccionario <i>dict</i>
2	<u>dict.copy ()</u> Devuelve una copia superficial de diccionario <i>dict</i>
3	<u>dict.fromkeys ()</u> Cree un nuevo diccionario con claves de seq y valores <i>establecidos</i> en <i>value</i> .
4 4	<u>dict.get (clave, predeterminado = Ninguno)</u> Para <i>clave</i> clave, devuelve valor o valor predeterminado si la clave no está en el diccionario
5 5	<u>dict.has_key (clave)</u> Devuelve <i>verdadero</i> si la clave en el diccionario <i>dict</i> , <i>falso</i> de lo contrario
6 6	<u>dict.items ()</u> Devuelve una lista de pares de tuplas de <i>dict</i> (clave, valor)
7 7	<u>dict.keys ()</u> Devuelve la lista de claves del diccionario <i>dict</i>
8	<u>dict.setdefault (clave, predeterminado = Ninguno)</u> Similar a <i>get ()</i> , pero establecerá <i>dict [clave] = predeterminado</i> si la <i>clave aún</i> no está en <i>dict</i>
9 9	<u>dict.update (dict2)</u>

	Agrega los <i>pares</i> clave-valor del diccionario <i>dict2</i> a <i>dict</i>
10	<code>dict.values ()</code> Devuelve la lista de los valores del diccionario <i>dict</i>

Python - Fecha y hora

Un programa Python puede manejar la fecha y la hora de varias maneras. La conversión entre formatos de fecha es una tarea común para las computadoras. Los módulos de tiempo y calendario de Python ayudan a rastrear fechas y horas.

¿Qué es Tick?

Los intervalos de tiempo son números de coma flotante en unidades de segundos. Los instantes particulares en el tiempo se expresan en segundos desde las 12:00 a.m., 1 de enero de 1970 (época).

Hay un módulo de **tiempo** popular disponible en Python que proporciona funciones para trabajar con tiempos y para convertir entre representaciones. La función `time.time ()` devuelve la hora actual del sistema en ticks desde las 12:00 a.m., 1 de enero de 1970 (época).

Ejemplo

```
#!/usr/bin/python
import time; # This is required to include time module.

ticks = time.time()
print "Number of ticks since 12:00am, January 1, 1970:",
ticks
```

Esto produciría un resultado algo así:

```
Number of ticks since 12:00am, January 1, 1970: 7186862.73399
```

La aritmética de fechas es fácil de hacer con las garrapatas. Sin embargo, las fechas anteriores a la época no se pueden representar de esta forma. Las fechas en el futuro lejano tampoco se pueden representar de esta manera: el punto de corte es en algún momento en 2038 para UNIX y Windows.

¿Qué es TimeTuple?

Muchas de las funciones de tiempo de Python manejan el tiempo como una tupla de 9 números, como se muestra a continuación:

Índice	Campo	Valores
0 0	Año de 4 dígitos	2008
1	Mes	1 a 12
2	Día	1 a 31
3	Hora	0 a 23
4 4	Minuto	0 a 59
5 5	Segundo	0 a 61 (60 o 61 son segundos de salto)
6 6	Día de la semana	0 a 6 (0 es lunes)
7 7	Día del año	1 a 366 (día juliano)
8	Horario de verano	-1, 0, 1, -1 significa que la biblioteca determina el horario de verano

La tupla anterior es equivalente a la estructura **struct_time** . Esta estructura tiene los siguientes atributos:

Índice	Atributos	Valores
0 0	tm_year	2008
1	tm_mon	1 a 12
2	tm_mday	1 a 31

3	tm_hour	0 a 23
4 4	tm_min	0 a 59
5 5	tm_sec	0 a 61 (60 o 61 son segundos de salto)
6 6	tm_wday	0 a 6 (0 es lunes)
7 7	tm_yday	1 a 366 (día juliano)
8	tm_isdst	-1, 0, 1, -1 significa que la biblioteca determina el horario de verano

Obtener hora actual

Para traducir un instante de tiempo desde unos *segundos desde el valor* de punto flotante de época a una tupla de tiempo, pase el valor de punto flotante a una función (por ejemplo, `localtime`) que devuelve una tupla de tiempo con los nueve elementos válidos.

```
#!/usr/bin/python
import time;

localtime = time.localtime(time.time())
print "Local current time :", localtime
```

Esto produciría el siguiente resultado, que podría formatearse en cualquier otra forma presentable:

```
Local current time : time.struct_time(tm_year=2013, tm_mon=7,
tm_mday=17, tm_hour=21, tm_min=26, tm_sec=3, tm_wday=2,
tm_yday=198, tm_isdst=0)
```

Obtener hora formateada

Puede formatear en cualquier momento según sus requisitos, pero el método simple para obtener el tiempo en formato legible es `asctime ()` -

```
#!/usr/bin/python
import time;

localtime = time.asctime( time.localtime(time.time()) )
print "Local current time :", localtime
```

Esto produciría el siguiente resultado:

```
Local current time : Tue Jan 13 10:17:09 2009
```

Obteniendo calendario por un mes

El módulo de calendario ofrece una amplia gama de métodos para jugar con calendarios anuales y mensuales. Aquí, imprimimos un calendario para un mes determinado (enero de 2008):

```
#!/usr/bin/python
import calendar

cal = calendar.month(2008, 1)
print "Here is the calendar:"
print cal
```

Esto produciría el siguiente resultado:

```
Here is the calendar:
    January 2008
Mo Tu We Th Fr Sa Su
     1  2  3  4  5  6
  7  8  9 10 11 12 13
14 15 16 17 18 19 20
21 22 23 24 25 26 27
28 29 30 31
```

El módulo de *tiempo*

Hay un módulo de **tiempo** popular disponible en Python que proporciona funciones para trabajar con tiempos y para convertir entre representaciones. Aquí está la lista de todos los métodos disponibles:

No Señor.	Función con descripción
1	<u>time.altzone</u> El desplazamiento de la zona horaria local de DST, en segundos al oeste de UTC, si se define uno. Esto es negativo si la zona horaria local del horario de verano está al este de UTC (como en Europa occidental, incluido el Reino Unido). Solo use esto si la luz del día no es cero.
2	<u>time.asctime ([tupletime])</u> Acepta una tupla de tiempo y devuelve una cadena legible de 24 caracteres como 'Mar 11 de diciembre 18:07:14 2008'.
3	<u>time.clock ()</u>

	Devuelve el tiempo de CPU actual como un número de segundos de punto flotante. Para medir los costos computacionales de diferentes enfoques, el valor de <code>time.clock</code> es más útil que el de <code>time.time</code> ().
4 4	<u><code>time.ctime</code> ([segundos])</u> Como <code>asctime</code> (<code>localtime</code> (secs)) y sin argumentos es como <code>asctime</code> ()
5 5	<u><code>time.gmtime</code> ([segundos])</u> Acepta un instante expresado en segundos desde la época y devuelve una tupla de tiempo <code>t</code> con la hora UTC. Nota: <code>t.tm_isdst</code> siempre es 0
6 6	<u><code>time.localtime</code> ([segundos])</u> Acepta un instante expresado en segundos desde la época y devuelve una tupla de tiempo <code>t</code> con la hora local (<code>t.tm_isdst</code> es 0 o 1, dependiendo de si DST se aplica a segundos instantáneos según las reglas locales).
7 7	<u><code>time.mktime</code> (tupletime)</u> Acepta un instante expresado como una tupla de tiempo en la hora local y devuelve un valor de punto flotante con el instante expresado en segundos desde la época.
8	<u><code>time.sleep</code> (segundos)</u> Suspende el hilo de llamada durante segundos segundos.
9 9	<u><code>time.strftime</code> (fmt [, tupletime])</u> Acepta un instante expresado como una tupla de tiempo en la hora local y devuelve una cadena que representa el instante especificado por string <code>fmt</code> .
10	<u><code>time.strptime</code> (str, fmt = '% a% b% d% H:% M:% S% Y')</u> Analiza <code>str</code> de acuerdo con el formato de cadena <code>fmt</code> y devuelve el instante en formato de tupla de tiempo.
11	<u><code>tiempo tiempo</code>()</u> Devuelve el instante de tiempo actual, un número de segundos de coma flotante desde la época.
12	<u><code>time.tzset</code> ()</u> Restablece las reglas de conversión de tiempo utilizadas por las rutinas de la biblioteca. La variable de entorno <code>TZ</code> especifica cómo se hace esto.

Veamos brevemente las funciones:

A continuación se encuentran disponibles dos atributos importantes con el módulo de tiempo:

No Señor.	Atributo con Descripción
1	<p>time.timezone</p> <p>El atributo time.timezone es el desplazamiento en segundos de la zona horaria local (sin DST) desde UTC (> 0 en las Américas; <= 0 en la mayoría de Europa, Asia, África).</p>
2	<p>time.tzname</p> <p>El atributo time.tzname es un par de cadenas dependientes de la configuración regional, que son los nombres de la zona horaria local sin y con DST, respectivamente.</p>

El módulo de *calendario*

El módulo de calendario proporciona funciones relacionadas con el calendario, incluidas las funciones para imprimir un calendario de texto para un mes o año determinado.

Por defecto, el calendario toma el lunes como el primer día de la semana y el domingo como el último. Para cambiar esto, llame a la función `calendar.setfirstweekday ()`.

Aquí hay una lista de funciones disponibles con el módulo de *calendario* :

No Señor.	Función con descripción
1	<p>calendar.calendar (año, w = 2, l = 1, c = 6)</p> <p>Devuelve una cadena multilínea con un calendario para el año año formateado en tres columnas separadas por espacios en c. w es el ancho en caracteres de cada fecha; cada línea tiene una longitud de $21 * w + 18 + 2 * c$. l es el número de líneas para cada semana.</p>
2	<p>calendar.firstweekday ()</p> <p>Devuelve la configuración actual para el día de la semana que comienza cada semana. Por defecto, cuando el calendario se importa por primera vez, es 0, es decir, lunes.</p>
3	<p>calendar.isleap (año)</p> <p>Devuelve True si el año es bisiesto; de lo contrario, falso.</p>

4 4	<p>calendar.leapdays (y1, y2)</p> <p>Devuelve el número total de días bisiestos en los años dentro del rango (y1, y2).</p>
5 5	<p>calendar.month (año, mes, w = 2, l = 1)</p> <p>Devuelve una cadena multilínea con un calendario para mes mes del año año, una línea por semana más dos líneas de encabezado. w es el ancho en caracteres de cada fecha; cada línea tiene una longitud de $7 * w + 6$. l es el número de líneas para cada semana.</p>
6 6	<p>calendar.monthcalendar (año, mes)</p> <p>Devuelve una lista de listas de entradas. Cada sublista denota una semana. Los días fuera del mes mes del año año se establecen en 0; los días dentro del mes se establecen en su día del mes, 1 y superior.</p>
7 7	<p>calendar.monthrange (año, mes)</p> <p>Devuelve dos enteros. El primero es el código del día de la semana para el primer día del mes mes en el año año; el segundo es el número de días en el mes. Los códigos de lunes a viernes son de 0 (lunes) a 6 (domingo); los números de mes son del 1 al 12.</p>
8	<p>calendar.prcal (año, w = 2, l = 1, c = 6)</p> <p>Como imprimir calendar.calendar (año, w, l, c).</p>
9 9	<p>calendar.prmonth (año, mes, w = 2, l = 1)</p> <p>Como imprimir calendar.month (año, mes, w, l).</p>
10	<p>calendar.setfirstweekday (día de la semana)</p> <p>Establece el primer día de cada semana en el código del día de la semana día de la semana. Los códigos de lunes a viernes son de 0 (lunes) a 6 (domingo).</p>
11	<p>calendar.timegm (tupletime)</p> <p>El inverso de time.gmtime: acepta un instante de tiempo en forma de tupla de tiempo y devuelve el mismo instante como un número de segundos de punto flotante desde la época.</p>
12	<p>calendar.weekday (año, mes, día)</p> <p>Devuelve el código del día de la semana para la fecha dada. Los códigos de lunes a viernes son de 0 (lunes) a 6 (domingo); los números de mes son 1 (enero) a 12 (diciembre).</p>

Otros módulos y funciones

Si está interesado, aquí encontrará una lista de otros módulos y funciones importantes para jugar con fecha y hora en Python:

- [El módulo de *fecha y hora*](#)
- [El módulo *pytz*](#)
- [El módulo *dateutil*](#)

Python - Funciones

Una función es un bloque de código organizado y reutilizable que se utiliza para realizar una única acción relacionada. Las funciones proporcionan una mejor modularidad para su aplicación y un alto grado de reutilización de código.

Como ya sabe, Python le ofrece muchas funciones integradas como print (), etc., pero también puede crear sus propias funciones. Estas funciones se denominan *funciones definidas por el usuario*.

Definiendo una función

Puede definir funciones para proporcionar la funcionalidad requerida. Aquí hay reglas simples para definir una función en Python.

- Los bloques de funciones comienzan con la palabra clave **def** seguida del nombre de la función y paréntesis ().
- Cualquier parámetro o argumento de entrada debe colocarse entre paréntesis. También puede definir parámetros dentro de estos paréntesis.
- La primera declaración de una función puede ser una declaración opcional - la cadena de documentación de la función o *cadena de documentación* .
- El bloque de código dentro de cada función comienza con dos puntos (:) y está sangrado.
- La sentencia return [expresión] sale de una función, opcionalmente devuelve una expresión a la persona que llama. Una declaración de retorno sin argumentos es lo mismo que return None.

Sintaxis

```
def functionname( parameters ):  
    "function_docstring"  
    function_suite  
    return [expression]
```

De manera predeterminada, los parámetros tienen un comportamiento posicional y debe informarlos en el mismo orden en que se definieron.

Ejemplo

La siguiente función toma una cadena como parámetro de entrada y la imprime en la pantalla estándar.

```
def printme( str ):  
    "This prints a passed string into this function"  
    print str  
    return
```

Llamar a una función

La definición de una función solo le da un nombre, especifica los parámetros que se incluirán en la función y estructura los bloques de código.

Una vez que se finaliza la estructura básica de una función, puede ejecutarla llamándola desde otra función o directamente desde el indicador de Python. El siguiente es el ejemplo para llamar a la función `printme()`:

```
#!/usr/bin/python  
  
# Function definition is here  
def printme( str ):  
    "This prints a passed string into this function"  
    print str  
    return;  
  
# Now you can call printme function  
printme("I'm first call to user defined function!")  
printme("Again second call to the same function")
```

Cuando se ejecuta el código anterior, produce el siguiente resultado:

```
I'm first call to user defined function!  
Again second call to the same function
```

Pase por referencia vs valor

Todos los parámetros (argumentos) en el lenguaje Python se pasan por referencia. Significa que si cambia a qué se refiere un parámetro dentro de una función, el cambio también se refleja en la función de llamada. Por ejemplo

```
#!/usr/bin/python  
  
# Function definition is here  
def changeme( mylist ):  
    "This changes a passed list into this function"  
    mylist.append([1,2,3,4]);  
    print "Values inside the function: ", mylist  
    return  
  
# Now you can call changeme function
```

```
mylist = [10,20,30];
changeme( mylist );
print "Values outside the function: ", mylist
```

Aquí, mantenemos la referencia del objeto pasado y los valores anexos en el mismo objeto. Entonces, esto produciría el siguiente resultado:

```
Values inside the function: [10, 20, 30, [1, 2, 3, 4]]
Values outside the function: [10, 20, 30, [1, 2, 3, 4]]
```

Hay un ejemplo más en el que el argumento se pasa por referencia y la referencia se sobrescribe dentro de la función llamada.

```
#!/usr/bin/python

# Function definition is here
def changeme( mylist ):
    "This changes a passed list into this function"
    mylist = [1,2,3,4]; # This would assign new reference in
mylist
    print "Values inside the function: ", mylist
    return

# Now you can call changeme function
mylist = [10,20,30];
changeme( mylist );
print "Values outside the function: ", mylist
```

El parámetro *mylist* es local para la función *changeme*. Cambiar *mylist* dentro de la función no afecta a *mylist*. La función no logra nada y finalmente esto produciría el siguiente resultado:

```
Values inside the function: [1, 2, 3, 4]
Values outside the function: [10, 20, 30]
```

Argumentos de funciones

Puede llamar a una función utilizando los siguientes tipos de argumentos formales:

- Argumentos requeridos
- Argumentos de palabras clave
- Argumentos predeterminados
- Argumentos de longitud variable

Argumentos requeridos

Los argumentos obligatorios son los argumentos pasados a una función en el orden posicional correcto. Aquí, el número de argumentos en la llamada a la función debe coincidir exactamente con la definición de la función.

Para llamar a la función *printme* (), definitivamente debe pasar un argumento, de lo contrario, se produce un error de sintaxis de la siguiente manera:

```
#!/usr/bin/python

# Function definition is here
def printme( str ):
    "This prints a passed string into this function"
    print str
    return;

# Now you can call printme function
printme()
```

Cuando se ejecuta el código anterior, produce el siguiente resultado:

```
Traceback (most recent call last):
  File "test.py", line 11, in <module>
    printme();
TypeError: printme() takes exactly 1 argument (0 given)
```

Argumentos de palabras clave

Los argumentos de palabras clave están relacionados con las llamadas a funciones. Cuando utiliza argumentos de palabras clave en una llamada de función, la persona que llama identifica los argumentos por el nombre del parámetro.

Esto le permite omitir argumentos o desordenarlos porque el intérprete de Python puede usar las palabras clave proporcionadas para hacer coincidir los valores con los parámetros. También puede realizar llamadas de palabras clave a la función *printme* () de las siguientes maneras:

```
#!/usr/bin/python

# Function definition is here
def printme( str ):
    "This prints a passed string into this function"
    print str
    return;

# Now you can call printme function
printme( str = "My string")
```

Cuando se ejecuta el código anterior, produce el siguiente resultado:

```
My string
```

El siguiente ejemplo da una imagen más clara. Tenga en cuenta que el orden de los parámetros no importa.

```
#!/usr/bin/python
```

```

# Function definition is here
def printinfo( name, age ):
    "This prints a passed info into this function"
    print "Name: ", name
    print "Age ", age
    return;

# Now you can call printinfo function
printinfo( age=50, name="miki" )

```

Cuando se ejecuta el código anterior, produce el siguiente resultado:

```

Name: miki
Age  50

```

Argumentos predeterminados

Un argumento predeterminado es un argumento que asume un valor predeterminado si no se proporciona un valor en la llamada de función para ese argumento. El siguiente ejemplo da una idea sobre los argumentos predeterminados, imprime la edad predeterminada si no se pasa:

```

#!/usr/bin/python

# Function definition is here
def printinfo( name, age = 35 ):
    "This prints a passed info into this function"
    print "Name: ", name
    print "Age ", age
    return;

# Now you can call printinfo function
printinfo( age=50, name="miki" )
printinfo( name="miki" )

```

Cuando se ejecuta el código anterior, produce el siguiente resultado:

```

Name: miki
Age  50
Name: miki
Age  35

```

Argumentos de longitud variable

Es posible que necesite procesar una función para más argumentos de los que especificó al definir la función. Estos argumentos se denominan argumentos de *longitud variable* y no se nombran en la definición de la función, a diferencia de los argumentos obligatorios y predeterminados.

La sintaxis para una función con argumentos variables que no son palabras clave es esta:

```
def functionname([formal_args,] *var_args_tuple ):
    "function_docstring"
    function_suite
    return [expression]
```

Se coloca un asterisco (*) antes del nombre de la variable que contiene los valores de todos los argumentos de variables sin palabras clave. Esta tupla permanece vacía si no se especifican argumentos adicionales durante la llamada a la función. El siguiente es un ejemplo simple:

```
#!/usr/bin/python

# Function definition is here
def printinfo( arg1, *vartuple ):
    "This prints a variable passed arguments"
    print "Output is: "
    print arg1
    for var in vartuple:
        print var
    return;

# Now you can call printinfo function
printinfo( 10 )
printinfo( 70, 60, 50 )
```

Cuando se ejecuta el código anterior, produce el siguiente resultado:

```
Output is:
10
Output is:
70
60
50
```

Las funciones *anónimas*

Estas funciones se denominan anónimas porque no se declaran de la manera estándar utilizando la palabra clave *def*. Puede usar la palabra clave *lambda* para crear pequeñas funciones anónimas.

- Las formas Lambda pueden tomar cualquier número de argumentos, pero devuelven solo un valor en forma de expresión. No pueden contener comandos o múltiples expresiones.
- Una función anónima no puede ser una llamada directa a imprimir porque lambda requiere una expresión
- Las funciones de Lambda tienen su propio espacio de nombres local y no pueden acceder a variables que no sean las de su lista de parámetros y las del espacio de nombres global.
- Aunque parece que los lambda son una versión de una línea de una función, no son equivalentes a las declaraciones en línea en C o C ++, cuyo propósito es pasar la asignación de la pila de funciones durante la invocación por razones de rendimiento.

Sintaxis

La sintaxis de las funciones *lambda* contiene solo una declaración, que es la siguiente:

```
lambda [arg1 [,arg2,.....argn]]:expression
```

A continuación se muestra el ejemplo para mostrar cómo funciona la forma de función *lambda* :

```
#!/usr/bin/python

# Function definition is here
sum = lambda arg1, arg2: arg1 + arg2;

# Now you can call sum as a function
print "Value of total : ", sum( 10, 20 )
print "Value of total : ", sum( 20, 20 )
```

Cuando se ejecuta el código anterior, produce el siguiente resultado:

```
Value of total : 30
Value of total : 40
```

La declaración de *devolución*

La sentencia `return [expresión]` sale de una función, opcionalmente devuelve una expresión a la persona que llama. Una declaración de retorno sin argumentos es lo mismo que `return None`.

Todos los ejemplos anteriores no devuelven ningún valor. Puede devolver un valor de una función de la siguiente manera:

```
#!/usr/bin/python

# Function definition is here
def sum( arg1, arg2 ):
    # Add both the parameters and return them."
    total = arg1 + arg2
    print "Inside the function : ", total
    return total;

# Now you can call sum function
total = sum( 10, 20 );
print "Outside the function : ", total
```

Cuando se ejecuta el código anterior, produce el siguiente resultado:

```
Inside the function : 30
Outside the function : 30
```

Alcance de las variables

Es posible que no se pueda acceder a todas las variables de un programa en todas las ubicaciones de ese programa. Esto depende de dónde haya declarado una variable.

El alcance de una variable determina la parte del programa donde puede acceder a un identificador particular. Hay dos ámbitos básicos de variables en Python:

- Variables globales
- Variables locales

Variables globales vs. locales

Las variables que se definen dentro de un cuerpo de función tienen un alcance local, y las definidas afuera tienen un alcance global.

Esto significa que solo se puede acceder a las variables locales dentro de la función en la que se declaran, mientras que todas las funciones pueden acceder a las variables globales en todo el cuerpo del programa. Cuando llama a una función, las variables declaradas dentro de ella se ponen en alcance. El siguiente es un ejemplo simple:

```
#!/usr/bin/python

total = 0; # This is global variable.
# Function definition is here
def sum( arg1, arg2 ):
    # Add both the parameters and return them."
    total = arg1 + arg2; # Here total is local variable.
    print "Inside the function local total : ", total
    return total;

# Now you can call sum function
sum( 10, 20 );
print "Outside the function global total : ", total
```

Cuando se ejecuta el código anterior, produce el siguiente resultado:

```
Inside the function local total : 30
Outside the function global total : 0
```


Python - Módulos

Un módulo le permite organizar lógicamente su código de Python. Agrupar el código relacionado en un módulo hace que el código sea más fácil de entender y usar. Un módulo es un objeto de Python con atributos nombrados arbitrariamente que puede vincular y hacer referencia.

Simplemente, un módulo es un archivo que consta de código Python. Un módulo puede definir funciones, clases y variables. Un módulo también puede incluir código ejecutable.

Ejemplo

El código de Python para un módulo llamado *aname* normalmente reside en un archivo llamado *aname.py*. Aquí hay un ejemplo de un módulo simple, *support.py*

```
def print_func( par ):  
    print "Hello : ", par  
    return
```

La declaración de *importación*

Puede usar cualquier archivo fuente de Python como módulo ejecutando una declaración de importación en algún otro archivo fuente de Python. La *importación* tiene la siguiente sintaxis:

```
import module1[, module2[,... moduleN]
```

Cuando el intérprete encuentra una declaración de importación, importa el módulo si el módulo está presente en la ruta de búsqueda. Una ruta de búsqueda es una lista de directorios que el intérprete busca antes de importar un módulo. Por ejemplo, para importar el módulo *support.py*, debe colocar el siguiente comando en la parte superior del script:

```
#!/usr/bin/python  
  
# Import module support  
import support  
  
# Now you can call defined function that module as follows  
support.print_func("Zara")
```

Cuando se ejecuta el código anterior, produce el siguiente resultado:

```
Hello : Zara
```

Un módulo se carga solo una vez, independientemente de la cantidad de veces que se importe. Esto evita que la ejecución del módulo suceda una y otra vez si se producen múltiples importaciones.

La declaración *de ... import*

La declaración *from* de Python le permite importar atributos específicos de un módulo al espacio de nombres actual. La *importación from ...* tiene la siguiente sintaxis:

```
from modname import name1[, name2[, ... nameN]]
```

Por ejemplo, para importar la función `fibonacci` del módulo `fib`, use la siguiente instrucción:

```
from fib import fibonacci
```

Esta declaración no importa todo el módulo `fib` en el espacio de nombres actual; solo introduce el elemento `fibonacci` del módulo `fib` en la tabla de símbolos global del módulo de importación.

La declaración *de ... import **

También es posible importar todos los nombres de un módulo al espacio de nombres actual mediante la siguiente instrucción de importación:

```
from modname import *
```

Esto proporciona una manera fácil de importar todos los elementos de un módulo al espacio de nombres actual; sin embargo, esta declaración debe usarse con moderación.

Módulos de localización

Cuando importa un módulo, el intérprete de Python busca el módulo en las siguientes secuencias:

- El directorio actual.
- Si no se encuentra el módulo, Python busca cada directorio en la variable de shell `PYTHONPATH`.
- Si todo lo demás falla, Python verifica la ruta predeterminada. En UNIX, esta ruta predeterminada es normalmente `/usr/local/lib/python/`.

La ruta de búsqueda del módulo se almacena en el módulo del sistema `sys` como la variable `sys.path`. La variable `sys.path` contiene el directorio actual, `PYTHONPATH` y el valor predeterminado dependiente de la instalación.

La variable *PYTHONPATH*

`PYTHONPATH` es una variable de entorno, que consiste en una lista de directorios. La sintaxis de `PYTHONPATH` es la misma que la de la variable de shell `PATH`.

Aquí hay una `PYTHONPATH` típica de un sistema Windows:

```
set PYTHONPATH = c:\python20\lib;
```

Y aquí hay una `PYTHONPATH` típica de un sistema UNIX:

```
set PYTHONPATH = /usr/local/lib/python
```

Espacios de nombres y alcance

Las variables son nombres (identificadores) que se asignan a objetos. Un *espacio de nombres* es un diccionario de nombres de variables (claves) y sus objetos (valores) correspondientes.

Una declaración de Python puede acceder a variables en un *espacio de nombres local* y en el *espacio de nombres global*. Si una variable local y una global tienen el mismo nombre, la variable local sombrea la variable global.

Cada función tiene su propio espacio de nombres local. Los métodos de clase siguen la misma regla de alcance que las funciones ordinarias.

Python hace suposiciones educadas sobre si las variables son locales o globales. Se supone que cualquier variable asignada a un valor en una función es local.

Por lo tanto, para asignar un valor a una variable global dentro de una función, primero debe usar la instrucción `global`.

La declaración `global VarName` le dice a Python que `VarName` es una variable global. Python deja de buscar la variable en el espacio de nombres local.

Por ejemplo, definimos una variable `Money` en el espacio de nombres global. Dentro de la función `Money`, asignamos un valor a `Money`, por lo tanto, Python asume `Money` como una variable local. Sin embargo, accedimos al valor de la variable local `Money` antes de configurarlo, por lo que el resultado es `UnboundLocalError`. Descomentar la declaración global soluciona el problema.

```
#!/usr/bin/python

Money = 2000
def AddMoney():
    # Uncomment the following line to fix the code:
    # global Money
    Money = Money + 1

print Money
AddMoney()
print Money
```

La función `dir ()`

La función incorporada `dir ()` devuelve una lista ordenada de cadenas que contienen los nombres definidos por un módulo.

La lista contiene los nombres de todos los módulos, variables y funciones que se definen en un módulo. El siguiente es un ejemplo simple:

```
#!/usr/bin/python
```

```
# Import built-in module math
import math

content = dir(math)
print content
```

Cuando se ejecuta el código anterior, produce el siguiente resultado:

```
['__doc__', '__file__', '__name__', 'acos', 'asin', 'atan',
'atan2', 'ceil', 'cos', 'cosh', 'degrees', 'e', 'exp',
'fabs', 'floor', 'fmod', 'frexp', 'hypot', 'ldexp', 'log',
'log10', 'modf', 'pi', 'pow', 'radians', 'sin', 'sinh',
'sqrt', 'tan', 'tanh']
```

Aquí, la variable de cadena especial `__name__` es el nombre del módulo y `__file__` es el nombre de archivo desde el que se cargó el módulo.

Las funciones *globales ()* y *locales ()*

Las funciones *globals ()* y *locals ()* se pueden usar para devolver los nombres en los espacios de nombres global y local dependiendo de la ubicación desde donde se llaman.

Si se llama a *locals ()* desde una función, devolverá todos los nombres a los que se puede acceder localmente desde esa función.

Si se llama a *globals ()* desde una función, devolverá todos los nombres a los que se puede acceder globalmente desde esa función.

El tipo de retorno de ambas funciones es diccionario. Por lo tanto, los nombres se pueden extraer utilizando la función de teclas *()*.

La *recarga ()* Función

Cuando el módulo se importa a un script, el código en la parte de nivel superior de un módulo se ejecuta solo una vez.

Por lo tanto, si desea volver a ejecutar el código de nivel superior en un módulo, puede usar la función *reload ()*. La función *reload ()* importa de nuevo un módulo previamente importado. La sintaxis de la función *reload ()* es esta:

```
reload(module_name)
```

Aquí, *module_name* es el nombre del módulo que desea volver a cargar y no la cadena que contiene el nombre del módulo. Por ejemplo, para volver a cargar el módulo *hello*, haga lo siguiente:

```
reload(hello)
```

Paquetes en Python

Un paquete es una estructura jerárquica de directorios de archivos que define un único entorno de aplicación Python que consta de módulos y subpaquetes y subpaquetes, y así sucesivamente.

Considere un archivo *Pots.py* disponible en el directorio *telefónico*. Este archivo tiene la siguiente línea de código fuente:

```
#!/usr/bin/python

def Pots():
    print "I'm Pots Phone"
```

De manera similar, tenemos otros dos archivos que tienen diferentes funciones con el mismo nombre que el anterior:

- Archivo *Phone / Isdn.py* que tiene la función *Isdn ()*
- Teléfono / archivo *G3.py* que tiene la función *G3 ()*

Ahora, cree un archivo más *__init__.py* en el directorio *telefónico* -

- Teléfono / *__init__.py*

Para que todas sus funciones estén disponibles cuando haya importado el teléfono, debe colocar declaraciones de importación explícitas en *__init__.py* de la siguiente manera:

```
from Pots import Pots
from Isdn import Isdn
from G3 import G3
```

Después de agregar estas líneas a *__init__.py*, tiene todas estas clases disponibles cuando importa el paquete del teléfono.

```
#!/usr/bin/python

# Now import your Phone Package.
import Phone

Phone.Pots()
Phone.Isdn()
Phone.G3()
```

Cuando se ejecuta el código anterior, produce el siguiente resultado:

```
I'm Pots Phone
I'm 3G Phone
I'm ISDN Phone
```

En el ejemplo anterior, hemos tomado un ejemplo de una sola función en cada archivo, pero puede mantener múltiples funciones en sus archivos. También puede definir diferentes clases de Python en esos archivos y luego puede crear sus paquetes a partir de esas clases.

Python - E / S de archivos

Este capítulo cubre todas las funciones básicas de E / S disponibles en Python. Para obtener más funciones, consulte la documentación estándar de Python.

Imprimir en la pantalla

La forma más sencilla de producir resultados es usar la instrucción *print* donde puede pasar cero o más expresiones separadas por comas. Esta función convierte las expresiones que pasa en una cadena y escribe el resultado en la salida estándar de la siguiente manera:

```
#!/usr/bin/python

print "Python is really a great language,", "isn't it?"
```

Esto produce el siguiente resultado en su pantalla estándar:

```
Python is really a great language, isn't it?
```

Lectura de entrada de teclado

Python proporciona dos funciones integradas para leer una línea de texto desde la entrada estándar, que por defecto proviene del teclado. Estas funciones son:

- Datos crudos
- entrada

La función *raw_input*

La función *raw_input* (*[prompt]*) lee una línea de la entrada estándar y la devuelve como una cadena (eliminando la nueva línea final).

```
#!/usr/bin/python

str = raw_input("Enter your input: ")
print "Received input is : ", str
```

Esto le pedirá que ingrese cualquier cadena y mostrará la misma cadena en la pantalla. Cuando escribí "¡Hola Python!", Su salida es así:

```
Enter your input: Hello Python
Received input is : Hello Python
```

La función de *entrada*

La función *input* (*[prompt]*) es equivalente a *raw_input*, excepto que supone que la entrada es una expresión Python válida y le devuelve el resultado evaluado.

```
#!/usr/bin/python
str = input("Enter your input: ")
print "Received input is : ", str
```

Esto produciría el siguiente resultado contra la entrada ingresada:

```
Enter your input: [x*5 for x in range(2,10,2)]
Received input is : [10, 20, 30, 40]
```

Abrir y cerrar archivos

Hasta ahora, ha estado leyendo y escribiendo en la entrada y salida estándar. Ahora, veremos cómo usar archivos de datos reales.

Python proporciona funciones y métodos básicos necesarios para manipular archivos de forma predeterminada. Puede hacer la mayor parte de la manipulación de archivos utilizando un objeto de **archivo** .

La función *abierto*

Antes de que pueda leer o escribir un archivo, debe abrirlo con la función *open* () incorporada de Python . Esta función crea un objeto de **archivo** , que se utilizaría para llamar a otros métodos de soporte asociados con él.

Sintaxis

```
file object = open(file_name [, access_mode] [, buffering])
```

Aquí están los detalles de los parámetros:

- **nombre_archivo** : el argumento *nombre_archivo* es un valor de cadena que contiene el nombre del archivo al que desea acceder.
- **access_mode** : el *access_mode* determina el modo en el que se debe abrir el archivo, es decir, leer, escribir, agregar, etc. A continuación se proporciona una lista completa de los valores posibles en la tabla. Este es un parámetro opcional y el modo de acceso al archivo predeterminado es *read* (r).
- **almacenamiento en búfer** : si el valor del almacenamiento en búfer se establece en 0, no se realiza el almacenamiento en búfer. Si el valor de almacenamiento en búfer es 1, el almacenamiento en línea se realiza al acceder a un archivo. Si especifica el valor de almacenamiento en búfer como un entero mayor que 1, la acción de almacenamiento en búfer se realiza con el tamaño de búfer indicado. Si es negativo, el tamaño del búfer es el valor predeterminado del sistema (comportamiento predeterminado).

Aquí hay una lista de los diferentes modos de abrir un archivo:

No Señor.	Modos y descripción
1	<p>r</p> <p>Abre un archivo para solo lectura. El puntero del archivo se coloca al principio del archivo. Este es el modo por defecto.</p>
2	<p>rb</p> <p>Abre un archivo para leer solo en formato binario. El puntero del archivo se coloca al principio del archivo. Este es el modo por defecto.</p>
3	<p>r +</p> <p>Abre un archivo para leer y escribir. El puntero del archivo colocado al comienzo del archivo.</p>
4 4	<p>rb +</p> <p>Abre un archivo para leer y escribir en formato binario. El puntero del archivo colocado al comienzo del archivo.</p>
5 5	<p>w</p> <p>Abre un archivo para escribir solo. Sobrescribe el archivo si el archivo existe. Si el archivo no existe, crea un nuevo archivo para escribir.</p>
6 6	<p>wb</p> <p>Abre un archivo para escribir solo en formato binario. Sobrescribe el archivo si el archivo existe. Si el archivo no existe, crea un nuevo archivo para escribir.</p>
7 7	<p>w +</p> <p>Abre un archivo para escribir y leer. Sobrescribe el archivo existente si el archivo existe. Si el archivo no existe, crea un nuevo archivo para leer y escribir.</p>
8	<p>wb +</p> <p>Abre un archivo para escribir y leer en formato binario. Sobrescribe el archivo existente si el archivo existe. Si el archivo no existe, crea un nuevo archivo para leer y escribir.</p>
9 9	<p>un</p> <p>Abre un archivo para anexar. El puntero del archivo está al final del archivo si el archivo existe. Es decir, el archivo está en el modo agregar. Si el archivo no</p>

	existe, crea un nuevo archivo para escribir.
10	<p>ab</p> <p>Abre un archivo para agregarlo en formato binario. El puntero del archivo está al final del archivo si el archivo existe. Es decir, el archivo está en el modo agregar. Si el archivo no existe, crea un nuevo archivo para escribir.</p>
11	<p>a +</p> <p>Abre un archivo para agregar y leer. El puntero del archivo está al final del archivo si el archivo existe. El archivo se abre en el modo agregar. Si el archivo no existe, crea un nuevo archivo para leer y escribir.</p>
12	<p>ab +</p> <p>Abre un archivo para agregar y leer en formato binario. El puntero del archivo está al final del archivo si el archivo existe. El archivo se abre en el modo agregar. Si el archivo no existe, crea un nuevo archivo para leer y escribir.</p>

El *archivo* Atributos del objeto

Una vez que se abre un archivo y tiene un objeto de *archivo*, puede obtener diversa información relacionada con ese archivo.

Aquí hay una lista de todos los atributos relacionados con el objeto de archivo:

No Señor.	Atributo y Descripción
1	<p>archivo.closed</p> <p>Devuelve verdadero si el archivo está cerrado, falso de lo contrario.</p>
2	<p>file.mode</p> <p>Devuelve el modo de acceso con el que se abrió el archivo.</p>
3	<p>nombre del archivo</p> <p>Devuelve el nombre del archivo.</p>
4 4	<p>file.softspace</p> <p>Devuelve falso si se requiere espacio explícitamente con print, verdadero de lo contrario.</p>

Ejemplo

```
#!/usr/bin/python

# Open a file
fo = open("foo.txt", "wb")
print "Name of the file: ", fo.name
print "Closed or not : ", fo.closed
print "Opening mode : ", fo.mode
print "Softspace flag : ", fo.softspace
```

Esto produce el siguiente resultado:

```
Name of the file:  foo.txt
Closed or not :   False
Opening mode :   wb
Softspace flag :  0
```

El método *close* ()

El método `close ()` de un objeto de *archivo* vacía cualquier información no escrita y cierra el objeto de archivo, después de lo cual no se puede escribir más.

Python cierra automáticamente un archivo cuando el objeto de referencia de un archivo se reasigna a otro archivo. Es una buena práctica usar el método `close ()` para cerrar un archivo.

Sintaxis

```
fileObject.close()
```

Ejemplo

```
#!/usr/bin/python

# Open a file
fo = open("foo.txt", "wb")
print "Name of the file: ", fo.name

# Close opened file
fo.close()
```

Esto produce el siguiente resultado:

```
Name of the file:  foo.txt
```

Leer y escribir archivos

El objeto de *archivo* proporciona un conjunto de métodos de acceso para hacernos la vida más fácil. Veríamos cómo usar los métodos *read ()* y *write ()* para leer y escribir archivos.

El método *write ()*

El método *write ()* escribe cualquier cadena en un archivo abierto. Es importante tener en cuenta que las cadenas de Python pueden tener datos binarios y no solo texto.

El método *write ()* no agrega un carácter de nueva línea ('\ n') al final de la cadena -

Sintaxis

```
fileObject.write(string)
```

Aquí, el parámetro pasado es el contenido que se escribirá en el archivo abierto.

Ejemplo

```
#!/usr/bin/python

# Open a file
fo = open("foo.txt", "wb")
fo.write( "Python is a great language.\nYeah its great!!\n")

# Close opened file
fo.close()
```

El método anterior crearía el archivo *foo.txt* y escribiría el contenido dado en ese archivo y finalmente cerraría ese archivo. Si abrieras este archivo, tendría el siguiente contenido.

```
Python is a great language.
Yeah its great!!
```

El método *read ()*

El método *read ()* lee una cadena de un archivo abierto. Es importante tener en cuenta que las cadenas de Python pueden tener datos binarios. aparte de los datos de texto.

Sintaxis

```
fileObject.read([count])
```

Aquí, el parámetro pasado es el número de bytes que se leerán del archivo abierto. Este método comienza a leer desde el principio del archivo y si falta el *recuento* , intenta leer lo más posible, tal vez hasta el final del archivo.

Ejemplo

Tomemos un archivo *foo.txt* , que creamos anteriormente.

```
#!/usr/bin/python

# Open a file
fo = open("foo.txt", "r+")
str = fo.read(10);
print "Read String is : ", str
# Close opened file
fo.close()
```

Esto produce el siguiente resultado:

```
Read String is : Python is
```

Posiciones de archivo

El método *tell* () le dice la posición actual dentro del archivo; en otras palabras, la próxima lectura o escritura se producirá en esa cantidad de bytes desde el comienzo del archivo.

El método de *búsqueda* (*desplazamiento* [, *desde*]) cambia la posición actual del archivo. El argumento de *desplazamiento* indica el número de bytes que se moverán. El argumento *from* especifica la posición de referencia desde donde se moverán los bytes.

Si *from* se establece en 0, significa usar el comienzo del archivo como la posición de referencia y 1 significa usar la posición actual como la posición de referencia y si se establece en 2, el final del archivo se tomará como la posición de referencia .

Ejemplo

Tomemos un archivo *foo.txt* , que creamos anteriormente.

```
#!/usr/bin/python

# Open a file
fo = open("foo.txt", "r+")
str = fo.read(10)
print "Read String is : ", str

# Check current position
position = fo.tell()
print "Current file position : ", position

# Reposition pointer at the beginning once again
```

```
position = fo.seek(0, 0);
str = fo.read(10)
print "Again read String is : ", str
# Close opened file
fo.close()
```

Esto produce el siguiente resultado:

```
Read String is : Python is
Current file position : 10
Again read String is : Python is
```

Renombrar y eliminar archivos

El módulo Python **os** proporciona métodos que lo ayudan a realizar operaciones de procesamiento de archivos, como renombrar y eliminar archivos.

Para usar este módulo, primero debe importarlo y luego puede llamar a cualquier función relacionada.

El método `rename ()`

El método `rename ()` toma dos argumentos, el nombre de archivo actual y el nuevo nombre de archivo.

Sintaxis

```
os.rename(current_file_name, new_file_name)
```

Ejemplo

El siguiente es el ejemplo para cambiar el nombre de un archivo existente `test1.txt` -

```
#!/usr/bin/python
import os

# Rename a file from test1.txt to test2.txt
os.rename( "test1.txt", "test2.txt" )
```

El método `remove ()`

Puede usar el método `remove ()` para eliminar archivos al proporcionar el nombre del archivo que se eliminará como argumento.

Sintaxis

```
os.remove(file_name)
```

Ejemplo

El siguiente es el ejemplo para eliminar un archivo existente *test2.txt* :

```
#!/usr/bin/python
import os

# Delete file test2.txt
os.remove("text2.txt")
```

Directorios en Python

Todos los archivos están contenidos en varios directorios, y Python no tiene problemas para manejarlos también. El módulo **os** tiene varios métodos que lo ayudan a crear, eliminar y cambiar directorios.

El método *mkdir ()*

Puede usar el método *mkdir ()* del módulo **os** para crear directorios en el directorio actual. Debe proporcionar un argumento para este método que contenga el nombre del directorio que se creará.

Sintaxis

```
os.mkdir("newdir")
```

Ejemplo

El siguiente es el ejemplo para crear una *prueba de* directorio en el directorio actual:

```
#!/usr/bin/python
import os

# Create a directory "test"
os.mkdir("test")
```

El método *chdir ()*

Puede usar el método *chdir ()* para cambiar el directorio actual. El método *chdir ()* toma un argumento, que es el nombre del directorio que desea que sea el directorio actual.

Sintaxis

```
os.chdir("newdir")
```

Ejemplo

El siguiente es el ejemplo para ir al directorio `"/home/newdir"`:

```
#!/usr/bin/python
import os

# Changing a directory to "/home/newdir"
os.chdir("/home/newdir")
```

El `getcwd ()` Método

El método `getcwd ()` muestra el directorio de trabajo actual.

Sintaxis

```
os.getcwd()
```

Ejemplo

El siguiente es el ejemplo para dar el directorio actual:

```
#!/usr/bin/python
import os

# This would give location of the current directory
os.getcwd()
```

El `rmdir ()` Método

El método `rmdir ()` elimina el directorio, que se pasa como argumento en el método.

Antes de eliminar un directorio, se debe eliminar todo su contenido.

Sintaxis

```
os.rmdir('dirname')
```

Ejemplo

El siguiente es el ejemplo para eliminar el directorio `"/tmp/test"`. Es necesario dar el nombre completo del directorio, de lo contrario buscaría ese directorio en el directorio actual.

```
#!/usr/bin/python
import os

# This would remove "/tmp/test" directory.
os.rmdir( "/tmp/test" )
```

Métodos relacionados con archivos y directorios

Hay tres fuentes importantes, que proporcionan una amplia gama de métodos de utilidad para manejar y manipular archivos y directorios en sistemas operativos Windows y Unix. Son los siguientes:

- Métodos de objeto de archivo : El objeto de *archivo* proporciona funciones para manipular archivos.
- Métodos de objetos del sistema operativo : proporciona métodos para procesar archivos y directorios.

Python - Manejo de excepciones

Python proporciona dos características muy importantes para manejar cualquier error inesperado en sus programas Python y para agregar capacidades de depuración en ellos:

- **Manejo de excepciones** : esto se tratará en este tutorial. Aquí hay una lista de Excepciones estándar disponibles en Python: Excepciones estándar .
- **Afirmaciones** : esto se trataría en el tutorial Afirmaciones en Python .

Lista de excepciones estándar -

No Señor.	Nombre de excepción y descripción
1	Excepción Clase base para todas las excepciones.
2	StopIteration Se genera cuando el método next () de un iterador no apunta a ningún objeto.
3	SystemExit Levantado por la función sys.exit ().
4 4	Error estándar Clase base para todas las excepciones integradas, excepto StopIteration y SystemExit.
5 5	ArithmeticError Clase base para todos los errores que ocurren para el cálculo numérico.
6 6	OverflowError

	Se genera cuando un cálculo excede el límite máximo para un tipo numérico.
7 7	FloatingPointError Se genera cuando falla un cálculo de coma flotante.
8	ZeroDivisionError Se genera cuando se realiza la división o módulo por cero para todos los tipos numéricos.
9 9	AserciónError Planteado en caso de falla de la declaración de Afirmación.
10	AttributeError Levantado en caso de falla de la referencia o asignación del atributo.
11	EOFError Se genera cuando no hay entrada desde la función <code>raw_input ()</code> o <code>input ()</code> y se alcanza el final del archivo.
12	ImportError Se genera cuando falla una declaración de importación.
13	Teclado interrumpido Se genera cuando el usuario interrumpe la ejecución del programa, generalmente presionando <code>Ctrl + c</code> .
14	LookupError Clase base para todos los errores de búsqueda.
15	IndexError Se genera cuando no se encuentra un índice en una secuencia.
dieciséis	KeyError Se genera cuando la clave especificada no se encuentra en el diccionario.
17	NameError Se genera cuando no se encuentra un identificador en el espacio de nombres

	local o global.
18 años	<p>UnboundLocalError</p> <p>Se genera cuando se intenta acceder a una variable local en una función o método pero no se le ha asignado ningún valor.</p>
19	<p>AmbienteError</p> <p>Clase base para todas las excepciones que se producen fuera del entorno de Python.</p>
20	<p>IOError</p> <p>Se genera cuando falla una operación de entrada / salida, como la instrucción print o la función open () cuando se intenta abrir un archivo que no existe.</p>
21	<p>IOError</p> <p>Generado para errores relacionados con el sistema operativo.</p>
22	<p>Error de sintaxis</p> <p>Se genera cuando hay un error en la sintaxis de Python.</p>
23	<p>IndentaciónError</p> <p>Levantado cuando la sangría no se especifica correctamente.</p>
24	<p>Error del sistema</p> <p>Se genera cuando el intérprete encuentra un problema interno, pero cuando se encuentra este error, el intérprete de Python no sale.</p>
25	<p>SystemExit</p> <p>Se genera cuando se cierra el intérprete de Python utilizando la función sys.exit (). Si no se maneja en el código, hace que el intérprete salga.</p>
26	<p>Error de teclado</p> <p>Se genera cuando se intenta una operación o función que no es válida para el tipo de datos especificado.</p>
27	<p>ValueError</p> <p>Se genera cuando la función incorporada para un tipo de datos tiene el tipo válido de argumentos, pero los argumentos tienen valores no válidos</p>

	especificados.
28	<p>Error de tiempo de ejecución</p> <p>Se genera cuando un error generado no cae en ninguna categoría.</p>
29	<p>NotImplementedError</p> <p>Se genera cuando un método abstracto que debe implementarse en una clase heredada no se implementa realmente.</p>

Afirmaciones en Python

Una afirmación es un control de cordura que puede activar o desactivar cuando haya terminado con las pruebas del programa.

La forma más fácil de pensar en una afirmación es compararla con una declaración **raise-if** (o para ser más exactos, una declaración raise-if-not). Se prueba una expresión y, si el resultado es falso, se genera una excepción.

Las afirmaciones se llevan a cabo mediante la declaración de aserción, la palabra clave más nueva para Python, introducida en la versión 1.5.

Los programadores a menudo colocan afirmaciones al comienzo de una función para verificar la entrada válida, y después de una llamada a la función para verificar la salida válida.

La declaración de *afirmación*

Cuando se encuentra con una declaración de aserción, Python evalúa la expresión que la acompaña, que es de esperar cierto. Si la expresión es falsa, Python genera una excepción *AssertionError*.

La sintaxis para afirmar es -

```
assert Expression[, Arguments]
```

Si la aserción falla, Python usa *ArgumentExpression* como argumento para *AssertionError*. Las excepciones *AssertionError* pueden detectarse y manejarse como cualquier otra excepción utilizando la instrucción try-except, pero si no se manejan, terminarán el programa y producirán un rastreo.

Ejemplo

Aquí hay una función que convierte una temperatura de grados Kelvin a grados Fahrenheit. Dado que cero grados Kelvin está tan frío como se pone, la función se rescata si ve una temperatura negativa:

```
#!/usr/bin/python
def KelvinToFahrenheit(Temperature):
```

```
    assert (Temperature >= 0), "Colder than absolute zero!"
    return ((Temperature-273)*1.8)+32
print KelvinToFahrenheit(273)
print int(KelvinToFahrenheit(505.78))
print KelvinToFahrenheit(-5)
```

Cuando se ejecuta el código anterior, produce el siguiente resultado:

```
32.0
451
Traceback (most recent call last):
File "test.py", line 9, in <module>
print KelvinToFahrenheit(-5)
File "test.py", line 4, in KelvinToFahrenheit
assert (Temperature >= 0), "Colder than absolute zero!"
AssertionError: Colder than absolute zero!
```

¿Qué es la excepción?

Una excepción es un evento que ocurre durante la ejecución de un programa que interrumpe el flujo normal de las instrucciones del programa. En general, cuando un script de Python se encuentra con una situación que no puede afrontar, genera una excepción. Una excepción es un objeto Python que representa un error.

Cuando una secuencia de comandos de Python genera una excepción, debe manejar la excepción de inmediato; de lo contrario, finaliza y se cierra.

Manejando una excepción

Si tiene algún código *sospechoso* que puede generar una excepción, puede defender su programa colocando el código sospechoso en un **intento**: bloquear. Después del `try`: block, incluya una declaración **except**:, seguida de un bloque de código que maneje el problema de la manera más elegante posible.

Sintaxis

Aquí hay una sintaxis simple de *try ... excepto ... else* blocks -

```
try:
    You do your operations here;
    .....
except ExceptionI:
    If there is ExceptionI, then execute this block.
except ExceptionII:
    If there is ExceptionII, then execute this block.
    .....
else:
    If there is no exception then execute this block.
```

Aquí hay algunos puntos importantes sobre la sintaxis mencionada anteriormente:

- Una sola declaración de prueba puede tener múltiples, excepto las declaraciones. Esto es útil cuando el bloque try contiene sentencias que pueden arrojar diferentes tipos de excepciones.
- También puede proporcionar una cláusula genérica excepto, que maneja cualquier excepción.
- Después de la (s) cláusula (s) excepto, puede incluir una cláusula else. El código en el bloque else se ejecuta si el código en el bloque try: no genera una excepción.
- El bloque else es un buen lugar para el código que no necesita probar: la protección del bloque.

Ejemplo

Este ejemplo abre un archivo, escribe contenido en el archivo y sale con gracia porque no hay ningún problema:

```
#!/usr/bin/python

try:
    fh = open("testfile", "w")
    fh.write("This is my test file for exception handling!!")
except IOError:
    print "Error: can't find file or read data"
else:
    print "Written content in the file successfully"
    fh.close()
```

Esto produce el siguiente resultado:

```
Written content in the file successfully
```

Ejemplo

Este ejemplo intenta abrir un archivo donde no tiene permiso de escritura, por lo que genera una excepción:

```
#!/usr/bin/python

try:
    fh = open("testfile", "r")
    fh.write("This is my test file for exception handling!!")
except IOError:
    print "Error: can't find file or read data"
else:
    print "Written content in the file successfully"
```

Esto produce el siguiente resultado:

```
Error: can't find file or read data
```

La cláusula *excepto* sin excepciones

También puede usar la instrucción `except` sin excepciones definidas de la siguiente manera:

```
try:
    You do your operations here;
    .....
except:
    If there is any exception, then execute this block.
    .....
else:
    If there is no exception then execute this block.
```

Este tipo de declaración **try-except** captura todas las excepciones que ocurren. Sin embargo, el uso de este tipo de declaración `try-except` no se considera una buena práctica de programación, ya que captura todas las excepciones pero no hace que el programador identifique la causa raíz del problema que puede ocurrir.

La cláusula *excepto* con múltiples excepciones

También puede usar la misma declaración *excepto* para manejar múltiples excepciones de la siguiente manera:

```
try:
    You do your operations here;
    .....
except(Exception1[, Exception2[,...ExceptionN]]):
    If there is any exception from the given exception list,
    then execute this block.
    .....
else:
    If there is no exception then execute this block.
```

La cláusula de intentar finalmente

Puede usar un bloque **finalmente**: junto con un bloque `try`:. El último bloque es un lugar para colocar cualquier código que deba ejecutarse, ya sea que el bloque de prueba provoque una excepción o no. La sintaxis de la declaración `try-finally` es esta:

```
try:
    You do your operations here;
    .....
    Due to any exception, this may be skipped.
finally:
    This would always be executed.
    .....
```

No puede usar la cláusula `else` también junto con una cláusula `finally`.

Ejemplo

```
#!/usr/bin/python

try:
    fh = open("testfile", "w")
    fh.write("This is my test file for exception handling!!")
finally:
    print "Error: can't find file or read data"
```

Si no tiene permiso para abrir el archivo en modo de escritura, esto producirá el siguiente resultado:

```
Error: can't find file or read data
```

El mismo ejemplo se puede escribir de manera más clara de la siguiente manera:

```
#!/usr/bin/python

try:
    fh = open("testfile", "w")
    try:
        fh.write("This is my test file for exception
handling!!")
    finally:
        print "Going to close the file"
        fh.close()
except IOError:
    print "Error: can't find file or read data"
```

Cuando se lanza una excepción en el bloque *try*, la ejecución pasa inmediatamente al bloque *finalmente*. Después de que se ejecutan todas las declaraciones en el bloque *finalmente*, la excepción se genera nuevamente y se maneja en las declaraciones *except*, si están presentes en la siguiente capa superior de la declaración *try-except*.

Argumento de una excepción

Una excepción puede tener un *argumento*, que es un valor que proporciona información adicional sobre el problema. El contenido del argumento varía según la excepción. Captura el argumento de una excepción al proporcionar una variable en la cláusula *except* de la siguiente manera:

```
try:
    You do your operations here;
    .....
except ExceptionType, Argument:
    You can print value of Argument here...
```

Si escribe el código para manejar una sola excepción, puede hacer que una variable siga el nombre de la excepción en la instrucción `except`. Si está atrapando varias excepciones, puede hacer que una variable siga la tupla de la excepción.

Esta variable recibe el valor de la excepción que contiene principalmente la causa de la excepción. La variable puede recibir un valor único o múltiples valores en forma de tupla. Esta tupla generalmente contiene la cadena de error, el número de error y una ubicación de error.

Ejemplo

El siguiente es un ejemplo para una sola excepción:

```
#!/usr/bin/python

# Define a function here.
def temp_convert(var):
    try:
        return int(var)
    except ValueError, Argument:
        print "The argument does not contain numbers\n",
Argument

# Call above function here.
temp_convert("xyz");
```

Esto produce el siguiente resultado:

```
The argument does not contain numbers
invalid literal for int() with base 10: 'xyz'
```

Levantando Excepciones

Puede generar excepciones de varias maneras utilizando la declaración de aumento. La sintaxis general para la declaración **raise** es la siguiente.

Sintaxis

```
raise [Exception [, args [, traceback]]]
```

Aquí, *Exception* es el tipo de excepción (por ejemplo, `NameError`) y el *argumento* es un valor para el argumento de excepción. El argumento es opcional; si no se proporciona, el argumento de excepción es Ninguno.

El argumento final, el rastreo, también es opcional (y rara vez se usa en la práctica), y si está presente, es el objeto de rastreo usado para la excepción.

Ejemplo

Una excepción puede ser una cadena, una clase o un objeto. La mayoría de las excepciones que plantea el núcleo de Python son clases, con un

argumento que es una instancia de la clase. Definir nuevas excepciones es bastante fácil y se puede hacer de la siguiente manera:

```
def functionName( level ):
    if level < 1:
        raise "Invalid level!", level
        # The code below to this would not be executed
        # if we raise the exception
```

Nota: Para capturar una excepción, una cláusula "excepto" debe referirse a la misma excepción lanzada, ya sea objeto de clase o cadena simple. Por ejemplo, para capturar la excepción anterior, debemos escribir la cláusula except de la siguiente manera:

```
try:
    Business Logic here...
except "Invalid level!":
    Exception handling here...
else:
    Rest of the code here...
```

Excepciones definidas por el usuario

Python también le permite crear sus propias excepciones derivando clases de las excepciones integradas estándar.

Aquí hay un ejemplo relacionado con *RuntimeError*. Aquí, se crea una clase que se subclasifica de *RuntimeError*. Esto es útil cuando necesita mostrar información más específica cuando se detecta una excepción.

En el bloque try, la excepción definida por el usuario se genera y se atrapa en el bloque except. La variable e se usa para crear una instancia de la clase *Networkerror*.

```
class Networkerror(RuntimeError):
    def __init__(self, arg):
        self.args = arg
```

Entonces, una vez que definió la clase anterior, puede generar la excepción de la siguiente manera:

```
try:
    raise Networkerror("Bad hostname")
except Networkerror,e:
    print e.args
```

Python - Orientado a objetos

Python ha sido un lenguaje orientado a objetos desde que existió. Debido a esto, crear y usar clases y objetos es francamente fácil. Este capítulo le ayuda a convertirse en un experto en el uso del soporte de programación orientado a objetos de Python.

Si no tiene experiencia previa con la programación orientada a objetos (OO), puede consultar un curso introductorio o al menos un tutorial de algún tipo para que tenga una idea de los conceptos básicos.

Sin embargo, aquí hay una pequeña introducción de la Programación Orientada a Objetos (OOP) para llevarlo a la velocidad:

Descripción general de la terminología de OOP

- **Clase** : un prototipo definido por el usuario para un objeto que define un conjunto de atributos que caracterizan cualquier objeto de la clase. Los atributos son miembros de datos (variables de clase y variables de instancia) y métodos, a los que se accede mediante notación de puntos.
- **Variable de clase** : una variable que comparten todas las instancias de una clase. Las variables de clase se definen dentro de una clase pero fuera de cualquiera de los métodos de la clase. Las variables de clase no se usan con tanta frecuencia como las variables de instancia.
- **Miembro de datos** : una variable de clase o variable de instancia que contiene datos asociados con una clase y sus objetos.
- **Sobrecarga de funciones** : la asignación de más de un comportamiento a una función en particular. La operación realizada varía según los tipos de objetos o argumentos involucrados.
- **Variable de instancia** : una variable que se define dentro de un método y pertenece solo a la instancia actual de una clase.
- **Herencia** : la transferencia de las características de una clase a otras clases que se derivan de ella.
- **Instancia** : un objeto individual de una determinada clase. Un objeto `obj` que pertenece a una clase `Circle`, por ejemplo, es una instancia de la clase `Circle`.
- **Instanciación** : la creación de una instancia de una clase.
- **Método** : un tipo especial de función que se define en una definición de clase.
- **Objeto** : una instancia única de una estructura de datos definida por su clase. Un objeto comprende miembros de datos (variables de clase y variables de instancia) y métodos.
- **Sobrecarga del operador** : la asignación de más de una función a un operador en particular.

Creando clases

La declaración de *clase* crea una nueva definición de clase. El nombre de la clase sigue inmediatamente a la *clase de* palabra clave seguida de dos puntos de la siguiente manera:

```
class ClassName:
    'Optional class documentation string'
    class_suite
```

- La clase tiene una cadena de documentación, a la que se puede acceder a través de *ClassName*.`__doc__`.
- El *class_suite* se compone de todos los componentes que definen declaraciones miembros de la clase, atributos de datos y funciones.

Ejemplo

El siguiente es el ejemplo de una clase simple de Python:

```
class Employee:
    'Common base class for all employees'
    empCount = 0

    def __init__(self, name, salary):
        self.name = name
        self.salary = salary
        Employee.empCount += 1

    def displayCount(self):
        print "Total Employee %d" % Employee.empCount

    def displayEmployee(self):
        print "Name : ", self.name, ", Salary: ", self.salary
```

- La variable *empCount* es una variable de clase cuyo valor se comparte entre todas las instancias de esta clase. Se puede acceder a este como *Employee.empCount* desde dentro de la clase o fuera de la clase.
- El primer método `__init__()` es un método especial, que se llama constructor de clases o método de inicialización que Python llama cuando crea una nueva instancia de esta clase.
- Usted declara otros métodos de clase como funciones normales con la excepción de que el primer argumento para cada método es *self*. Python agrega el argumento *propio* a la lista por usted; no necesita incluirlo cuando llame a los métodos.

Crear objetos de instancia

Para crear instancias de una clase, llame a la clase utilizando el nombre de la clase y pase los argumentos que *acepte* su método `__init__`.

```
"This would create first object of Employee class"
emp1 = Employee("Zara", 2000)
"This would create second object of Employee class"
emp2 = Employee("Manni", 5000)
```

Acceso a atributos

Accede a los atributos del objeto utilizando el operador de punto con objeto. Se accedería a la variable de clase usando el nombre de clase de la siguiente manera:

```
emp1.displayEmployee()  
emp2.displayEmployee()  
print "Total Employee %d" % Employee.empCount
```

Ahora, juntando todos los conceptos:

```
#!/usr/bin/python  
  
class Employee:  
    'Common base class for all employees'  
    empCount = 0  
  
    def __init__(self, name, salary):  
        self.name = name  
        self.salary = salary  
        Employee.empCount += 1  
  
    def displayCount(self):  
        print "Total Employee %d" % Employee.empCount  
  
    def displayEmployee(self):  
        print "Name : ", self.name, ", Salary: ", self.salary  
  
"This would create first object of Employee class"  
emp1 = Employee("Zara", 2000)  
"This would create second object of Employee class"  
emp2 = Employee("Manni", 5000)  
emp1.displayEmployee()  
emp2.displayEmployee()  
print "Total Employee %d" % Employee.empCount
```

Cuando se ejecuta el código anterior, produce el siguiente resultado:

```
Name : Zara ,Salary: 2000  
Name : Manni ,Salary: 5000  
Total Employee 2
```

Puede agregar, eliminar o modificar atributos de clases y objetos en cualquier momento:

```
emp1.age = 7 # Add an 'age' attribute.  
emp1.age = 8 # Modify 'age' attribute.  
del emp1.age # Delete 'age' attribute.
```

En lugar de usar las declaraciones normales para acceder a los atributos, puede usar las siguientes funciones:

- El **getattr (obj, nombre [, predeterminado])** - para acceder al atributo del objeto.
- El **hasattr (obj, nombre)** : para verificar si un atributo existe o no.
- El **setattr (obj, nombre, valor)** - para establecer un atributo. Si el atributo no existe, entonces se crearía.

- El **delattr (obj, nombre)** : para eliminar un atributo.

```
hasattr(emp1, 'age')    # Returns true if 'age' attribute
exists
getattr(emp1, 'age')   # Returns value of 'age' attribute
setattr(emp1, 'age', 8) # Set attribute 'age' at 8
delattr(emp1, 'age')   # Delete attribute 'age'
```

Atributos de clase incorporados

Cada clase de Python sigue los atributos incorporados y se puede acceder a ellos utilizando el operador de punto como cualquier otro atributo:

- **__dict__** - Diccionario que contiene el espacio de nombres de la clase.
- **__doc__** - Cadena de documentación de clase o ninguna, si no está definida.
- **__name__** - Nombre de la clase.
- **__module__** - Nombre del módulo en el que se define la clase. Este atributo es "__main__" en modo interactivo.
- **__bases__** : una tupla posiblemente vacía que contiene las clases base, en el orden en que aparecen en la lista de clases base.

Para la clase anterior intentemos acceder a todos estos atributos:

```
#!/usr/bin/python

class Employee:
    'Common base class for all employees'
    empCount = 0

    def __init__(self, name, salary):
        self.name = name
        self.salary = salary
        Employee.empCount += 1

    def displayCount(self):
        print "Total Employee %d" % Employee.empCount

    def displayEmployee(self):
        print "Name : ", self.name, ", Salary: ", self.salary

print "Employee.__doc__:", Employee.__doc__
print "Employee.__name__:", Employee.__name__
print "Employee.__module__:", Employee.__module__
print "Employee.__bases__:", Employee.__bases__
print "Employee.__dict__:", Employee.__dict__
```

Cuando se ejecuta el código anterior, produce el siguiente resultado:

```
Employee.__doc__: Common base class for all employees
Employee.__name__: Employee
Employee.__module__: __main__
Employee.__bases__: ()
```

```
Employee.__dict__: {'__module__': '__main__', 'displayCount':
<function displayCount at 0xb7c84994>, 'empCount': 2,
'displayEmployee': <function displayEmployee at 0xb7c8441c>,
'__doc__': 'Common base class for all employees',
'__init__': <function __init__ at 0xb7c846bc>}
```

Destrucción de objetos (recolección de basura)

Python elimina objetos innecesarios (tipos incorporados o instancias de clase) automáticamente para liberar espacio en la memoria. El proceso por el cual Python reclama periódicamente bloques de memoria que ya no están en uso se denomina recolección de basura.

El recolector de basura de Python se ejecuta durante la ejecución del programa y se activa cuando el recuento de referencia de un objeto llega a cero. El recuento de referencia de un objeto cambia a medida que cambia el número de alias que lo señalan.

El recuento de referencias de un objeto aumenta cuando se le asigna un nuevo nombre o se coloca en un contenedor (lista, tupla o diccionario). El recuento de referencias del objeto disminuye cuando se elimina con *del*, su referencia se reasigna o su referencia queda fuera de alcance. Cuando el recuento de referencia de un objeto llega a cero, Python lo recopila automáticamente.

```
a = 40          # Create object <40>
b = a          # Increase ref. count  of <40>
c = [b]        # Increase ref. count  of <40>

del a          # Decrease ref. count  of <40>
b = 100        # Decrease ref. count  of <40>
c[0] = -1      # Decrease ref. count  of <40>
```

Normalmente no se dará cuenta cuando el recolector de basura destruya una instancia huérfana y reclame su espacio. Pero una clase puede implementar el método especial `__del__()`, llamado destructor, que se invoca cuando la instancia está a punto de ser destruida. Este método puede usarse para limpiar cualquier recurso que no sea de memoria usado por una instancia.

Ejemplo

Este destructor `__del__()` imprime el nombre de la clase de una instancia que está a punto de ser destruida.

```
#!/usr/bin/python

class Point:
    def __init__( self, x=0, y=0):
        self.x = x
        self.y = y
    def __del__(self):
        class_name = self.__class__.__name__
```

```

        print class_name, "destroyed"

pt1 = Point()
pt2 = pt1
pt3 = pt1
print id(pt1), id(pt2), id(pt3) # prints the ids of the
objects
del pt1
del pt2
del pt3

```

Cuando se ejecuta el código anterior, produce el siguiente resultado:

```

3083401324 3083401324 3083401324
Point destroyed

```

Nota : Idealmente, debe definir sus clases en un archivo separado, luego debe importarlas en su archivo de programa principal utilizando la declaración de *importación* .

Herencia de clase

En lugar de comenzar desde cero, puede crear una clase derivándola de una clase preexistente enumerando la clase primaria entre paréntesis después del nuevo nombre de clase.

La clase secundaria hereda los atributos de su clase primaria, y puede usar esos atributos como si estuvieran definidos en la clase secundaria. Una clase secundaria también puede anular los miembros y métodos de datos del padre.

Sintaxis

Las clases derivadas se declaran como su clase principal; sin embargo, se proporciona una lista de clases base para heredar después del nombre de la clase:

```

class SubClassName (ParentClass1[, ParentClass2, ...]):
    'Optional class documentation string'
    class_suite

```

Ejemplo

```

#!/usr/bin/python

class Parent:          # define parent class
    parentAttr = 100
    def __init__(self):
        print "Calling parent constructor"

    def parentMethod(self):
        print 'Calling parent method'

```

```

def setAttr(self, attr):
    Parent.parentAttr = attr

def getAttr(self):
    print "Parent attribute :", Parent.parentAttr

class Child(Parent): # define child class
    def __init__(self):
        print "Calling child constructor"

    def childMethod(self):
        print 'Calling child method'

c = Child()          # instance of child
c.childMethod()     # child calls its method
c.parentMethod()    # calls parent's method
c.setAttr(200)      # again call parent's method
c.getAttr()         # again call parent's method

```

Cuando se ejecuta el código anterior, produce el siguiente resultado:

```

Calling child constructor
Calling child method
Calling parent method
Parent attribute : 200

```

De manera similar, puede conducir una clase desde varias clases principales de la siguiente manera:

```

class A:          # define your class A
    .....

class B:          # define your class B
    .....

class C(A, B):    # subclass of A and B
    .....

```

Puede usar las funciones `issubclass ()` o `isinstance ()` para verificar las relaciones de dos clases e instancias.

- La función booleana **issubclass (sub, sup)** devuelve verdadero si la subclase dada **sub** es de hecho una subclase de la superclase **sup** .
- La función booleana **isinstance (obj, Class)** devuelve verdadero si *obj* es una instancia de la clase *Class* o es una instancia de una subclase de *Class*

Métodos de anulación

Siempre puede anular sus métodos de clase principal. Una razón para anular los métodos de los padres es porque es posible que desee una funcionalidad especial o diferente en su subclase.

Ejemplo

```
#!/usr/bin/python

class Parent:          # define parent class
    def myMethod(self):
        print 'Calling parent method'

class Child(Parent):  # define child class
    def myMethod(self):
        print 'Calling child method'

c = Child()           # instance of child
c.myMethod()         # child calls overridden method
```

Cuando se ejecuta el código anterior, produce el siguiente resultado:

```
Calling child method
```

Métodos básicos de sobrecarga

La siguiente tabla enumera algunas funciones genéricas que puede anular en sus propias clases:

No Señor.	Método, descripción y muestra de llamada
1	<code>__init__ (self [, args ...])</code> Constructor (con cualquier argumento opcional) Llamada de muestra: <i>obj = className (args)</i>
2	<code>__del__ (auto)</code> Destructor, elimina un objeto Llamada de muestra: <i>del obj</i>
3	<code>__repr__ (auto)</code> Representación de cadena evaluable

	Llamada de muestra: <i>repr (obj)</i>
4 4	<code>__str__ (auto)</code> Representación de cadena imprimible Llamada de muestra: <i>str (obj)</i>
5 5	<code>__cmp__ (auto, x)</code> Comparación de objetos Llamada de muestra: <i>cmp (obj, x)</i>

Operadores de sobrecarga

Supongamos que ha creado una clase `Vector` para representar vectores bidimensionales, ¿qué sucede cuando usa el operador más para agregarlos? Lo más probable es que Python te grite.

Sin embargo, podría definir el método `__add__` en su clase para realizar la suma de vectores y luego el operador más se comportaría según las expectativas:

Ejemplo

```
#!/usr/bin/python

class Vector:
    def __init__(self, a, b):
        self.a = a
        self.b = b

    def __str__(self):
        return 'Vector (%d, %d)' % (self.a, self.b)

    def __add__(self, other):
        return Vector(self.a + other.a, self.b + other.b)

v1 = Vector(2,10)
v2 = Vector(5,-2)
print v1 + v2
```

Cuando se ejecuta el código anterior, produce el siguiente resultado:

```
Vector(7,8)
```

Ocultar datos

Los atributos de un objeto pueden o no ser visibles fuera de la definición de clase. Debe nombrar los atributos con un doble prefijo de subrayado, y esos atributos no serán directamente visibles para los extraños.

Ejemplo

```
#!/usr/bin/python

class JustCounter:
    __secretCount = 0

    def count(self):
        self.__secretCount += 1
        print self.__secretCount

counter = JustCounter()
counter.count()
counter.count()
print counter.__secretCount
```

Cuando se ejecuta el código anterior, produce el siguiente resultado:

```
1
2
Traceback (most recent call last):
  File "test.py", line 12, in <module>
    print counter.__secretCount
AttributeError: JustCounter instance has no attribute
'__secretCount'
```

Python protege a esos miembros cambiando internamente el nombre para incluir el nombre de la clase. Puede acceder a atributos como *object._className__attrName*. Si reemplaza su última línea de la siguiente manera, entonces funciona para usted:

```
.....
print counter._JustCounter__secretCount
```

Cuando se ejecuta el código anterior, produce el siguiente resultado:

```
1
2
2
```

Python - Expresiones regulares

Una *expresión regular* es una secuencia especial de caracteres que le ayuda a unir o encontrar otras cadenas o conjuntos de cadenas, utilizando una sintaxis especializada contenida en un patrón. Las expresiones regulares son ampliamente utilizadas en el mundo UNIX.

El módulo **re** proporciona soporte completo para expresiones regulares similares a Perl en Python. El módulo **re** genera la excepción **re.error** si se produce un error al compilar o utilizar una expresión regular.

Cubriremos dos funciones importantes, que se utilizarían para manejar expresiones regulares. Pero una pequeña cosa primero: hay varios caracteres, que tendrían un significado especial cuando se usan en expresiones regulares. Para evitar cualquier confusión al tratar con expresiones regulares, **usaríamos** cadenas sin procesar como **r'expression'**.

La función de *coincidencia*

Esta función intenta hacer coincidir el *patrón* RE con la *cadena* con *banderas* opcionales.

Aquí está la sintaxis para esta función:

```
re.match(pattern, string, flags=0)
```

Aquí está la descripción de los parámetros:

No Señor.	Descripción de parámetros
1	modelo Esta es la expresión regular que debe coincidir.
2	cuerda Esta es la cadena, que se buscaría para que coincida con el patrón al comienzo de la cadena.
3	banderas Puede especificar diferentes indicadores utilizando OR a nivel de bit (). Estos son modificadores, que se enumeran en la tabla a continuación.

La función **re.match** devuelve un objeto de **coincidencia** en caso de éxito, **Ninguno** en caso de error. Usamos la función **group (num)** o **groups ()** del objeto de **coincidencia** para obtener una expresión coincidente.

No	Método de objeto de coincidencia y descripción
----	--

Señor.	
1	<p>grupo (num = 0)</p> <p>Este método devuelve una coincidencia completa (o un número de subgrupo específico)</p>
2	<p>grupos ()</p> <p>Este método devuelve todos los subgrupos coincidentes en una tupla (vacía si no hubiera ninguna)</p>

Ejemplo

```
#!/usr/bin/python
import re

line = "Cats are smarter than dogs"

matchObj = re.match( r'(.*) are (.*) .*', line, re.M|re.I)

if matchObj:
    print "matchObj.group() : ", matchObj.group()
    print "matchObj.group(1) : ", matchObj.group(1)
    print "matchObj.group(2) : ", matchObj.group(2)
else:
    print "No match!!"
```

Cuando se ejecuta el código anterior, produce el siguiente resultado:

```
matchObj.group() : Cats are smarter than dogs
matchObj.group(1) : Cats
matchObj.group(2) : smarter
```

La función de *búsqueda*

Esta función busca la primera aparición del *patrón* RE dentro de una *cadena* con *banderas* opcionales .

Aquí está la sintaxis para esta función:

```
re.search(pattern, string, flags=0)
```

Aquí está la descripción de los parámetros:

No Señor.	Descripción de parámetros
-----------	---------------------------

1	<p>modelo</p> <p>Esta es la expresión regular que debe coincidir.</p>
2	<p>cuerda</p> <p>Esta es la cadena, que se buscaría para que coincidiera con el patrón en cualquier parte de la cadena.</p>
3	<p>banderas</p> <p>Puede especificar diferentes indicadores utilizando OR a nivel de bit (). Estos son modificadores, que se enumeran en la tabla a continuación.</p>

La función `re.search` devuelve un objeto de **coincidencia** en caso de éxito, **ninguno** en caso de error. Usamos la función `group (num)` o `groups ()` del objeto de **coincidencia** para obtener una expresión coincidente.

No Señor.	Métodos de objetos de coincidencia y descripción
1	<p>grupo (num = 0)</p> <p>Este método devuelve una coincidencia completa (o un número de subgrupo específico)</p>
2	<p>grupos ()</p> <p>Este método devuelve todos los subgrupos coincidentes en una tupla (vacía si no hubiera ninguna)</p>

Ejemplo

```
#!/usr/bin/python
import re

line = "Cats are smarter than dogs";

searchObj = re.search( r'(.*) are (.*) .*', line, re.M|re.I)

if searchObj:
    print "searchObj.group() : ", searchObj.group()
    print "searchObj.group(1) : ", searchObj.group(1)
    print "searchObj.group(2) : ", searchObj.group(2)
else:
    print "Nothing found!!"
```

Cuando se ejecuta el código anterior, produce el siguiente resultado:

```
searchObj.group() : Cats are smarter than dogs
searchObj.group(1) : Cats
searchObj.group(2) : smarter
```

Coincidencia versus búsqueda

Python ofrece dos operaciones primitivas diferentes basadas en expresiones regulares: las comprobaciones de **coincidencia** para una coincidencia solo al comienzo de la cadena, mientras que la **búsqueda** comprueba una coincidencia en cualquier parte de la cadena (esto es lo que hace Perl por defecto).

Ejemplo

```
#!/usr/bin/python
import re

line = "Cats are smarter than dogs";

matchObj = re.match( r'dogs', line, re.M|re.I)
if matchObj:
    print "match --> matchObj.group() : ", matchObj.group()
else:
    print "No match!!"

searchObj = re.search( r'dogs', line, re.M|re.I)
if searchObj:
    print "search --> searchObj.group() : ", searchObj.group()
else:
    print "Nothing found!!"
```

Cuando se ejecuta el código anterior, produce el siguiente resultado:

```
No match!!
search --> searchObj.group() : dogs
```

Buscar y reemplazar

Uno de los métodos **re** más importantes que usan expresiones regulares es **sub** .

Sintaxis

```
re.sub(pattern, repl, string, max=0)
```

Este método reemplaza todas las ocurrencias del *patrón* RE en *cadena* con *repl*, sustituyendo todas las ocurrencias a menos que se proporcione *max*. Este método devuelve una cadena modificada.

Ejemplo

```
#!/usr/bin/python
import re

phone = "2004-959-559 # This is Phone Number"

# Delete Python-style comments
num = re.sub(r'#.*$', "", phone)
print "Phone Num : ", num

# Remove anything other than digits
num = re.sub(r'\D', "", phone)
print "Phone Num : ", num
```

Cuando se ejecuta el código anterior, produce el siguiente resultado:

```
Phone Num : 2004-959-559
```

```
Phone Num : 2004959559
```

Modificadores de Expresión Regular: Banderas Opcionales

Los literales de expresión regular pueden incluir un modificador opcional para controlar varios aspectos de la coincidencia. Los modificadores se especifican como una bandera opcional. Puede proporcionar múltiples modificadores usando OR (|) exclusivo, como se muestra anteriormente y puede estar representado por uno de estos:

No Señor.	Modificador y Descripción
1	re.I Realiza coincidencias entre mayúsculas y minúsculas.
2	re.L Interpreta palabras de acuerdo con el entorno local actual. Esta interpretación afecta al grupo alfabético (<code>\w</code> y <code>\W</code>), así como al comportamiento de los límites de las palabras (<code>\b</code> y <code>\B</code>).
3	movimiento rápido del ojo Hace que <code>\$</code> coincida con el final de una línea (no solo con el final de la cadena) y hace que <code>^</code> coincida con el inicio de cualquier línea (no solo con el inicio de la cadena).

4 4	re.S Hace que un punto (punto) coincida con cualquier carácter, incluida una nueva línea.
5 5	re.U Interpreta letras de acuerdo con el juego de caracteres Unicode. Este indicador afecta el comportamiento de <code>\w</code> , <code>\W</code> , <code>\b</code> , <code>\B</code> .
6 6	re.X Permite la sintaxis de expresión regular "más linda". Ignora los espacios en blanco (excepto dentro de un conjunto <code>[]</code> o cuando se escapa por una barra invertida) y trata el <code>#</code> sin escape como un marcador de comentario.

Patrones de expresión regular

A excepción de los caracteres de control, `(+?. * ^ $ () [] {} | \)`, Todos los caracteres coinciden. Puede escapar de un personaje de control precediéndolo con una barra invertida.

La siguiente tabla enumera la sintaxis de expresión regular que está disponible en Python:

No Señor.	Patrón y descripción
1	^ Coincide con el comienzo de la línea.
2	\$ Coincide con el final de línea.
3	. Coincide con cualquier carácter, excepto la nueva línea. Usar la opción <code>m</code> también le permite coincidir con la nueva línea.
4 4	[...] Coincide con cualquier carácter entre paréntesis.
5 5	[^ ...]

	<p>Coincide con cualquier carácter individual que no esté entre paréntesis</p>
6 6	<p>re*</p> <p>Coincide con 0 o más ocurrencias de la expresión anterior.</p>
7 7	<p>re +</p> <p>Coincide con 1 o más ocurrencias de la expresión anterior.</p>
8	<p>¿re?</p> <p>Coincide con 0 o 1 aparición de la expresión anterior.</p>
9 9	<p>re {n}</p> <p>Coincide exactamente n número de ocurrencias de expresiones anteriores.</p>
10	<p>re {n,}</p> <p>Coincide con n o más ocurrencias de expresiones anteriores.</p>
11	<p>re {n, m}</p> <p>Coincide al menos n y como máximo m ocurrencias de expresiones anteriores.</p>
12	<p>a si</p> <p>Coincide con a o b.</p>
13	<p>(re)</p> <p>Agrupar expresiones regulares y recordar texto coincidente.</p>
14	<p>(? imx)</p> <p>Alterna temporalmente en las opciones i, m o x dentro de una expresión regular. Si entre paréntesis, solo esa área se ve afectada.</p>
15	<p>(? -imx)</p> <p>Desactiva temporalmente las opciones i, m o x dentro de una expresión regular. Si entre paréntesis, solo esa área se ve afectada.</p>
dieciséis	<p>(?: re)</p> <p>Agrupar expresiones regulares sin recordar texto coincidente.</p>

17	(? imx: re) Alterna temporalmente entre las opciones i, m o x entre paréntesis.
18 años	(? -imx: re) Desactiva temporalmente las opciones i, m o x entre paréntesis.
19	(? # ...) Comentario.
20	(? = re) Especifica la posición usando un patrón. No tiene rango.
21	(?! re) Especifica la posición usando la negación del patrón. No tiene rango.
22	(?> re) Coincide con el patrón independiente sin retroceso.
23	\w Coincide con caracteres de palabras.
24	\W Coincide con caracteres que no son palabras.
25	\s Coincide con espacios en blanco. Equivalente a <code>[\t\n\r\f]</code> .
26	\S Coincide con espacios no en blanco.
27	\d Coincide con dígitos. Equivalente a <code>[0-9]</code> .
28	\D Partidos no digitales.

29	\UN Coincide con el comienzo de la cadena.
30	\Z Coincide con el final de la cadena. Si existe una nueva línea, coincide justo antes de la nueva línea.
31	\z Coincide con el final de la cadena.
32	\SOL Los partidos señalan dónde terminó el último partido.
33	\si Coincide con los límites de palabras cuando está fuera de los corchetes. Coincide con el espacio de retroceso (0x08) dentro de los corchetes.
34	\SI Coincide con los límites sin palabras.
35	\n, \t, etc. Coincide con nuevas líneas, retornos de carro, pestañas, etc.
36	\1 ... \9 Coincide enésima subexpresión agrupada.
37	\10 Coincide con la subexpresión agrupada enésima si ya coincide. De lo contrario, se refiere a la representación octal de un código de caracteres.

Ejemplos de expresiones regulares

Personajes literales

No Señor.	Ejemplo y descripción
-----------	-----------------------

1	pitón Coincide con "pitón".
---	---------------------------------------

Clases de personajes

No Señor.	Ejemplo y descripción
1	[Pp] ython Haga coincidir "Python" o "python"
2	frotar [ye] Haga coincidir "ruby" o "rube"
3	[aeiou] Emparejar cualquier vocal minúscula
4 4	[0-9] Coincide con cualquier dígito; igual que [0123456789]
5 5	[Arizona] Coincide con cualquier letra minúscula ASCII
6 6	[ARIZONA] Coincide con cualquier letra ASCII mayúscula
7 7	[a-zA-Z0-9] Coincidir con cualquiera de los anteriores
8	[^ aeiou] Empareja cualquier cosa que no sea una vocal en minúsculas
9 9	[^ 0-9] Iguala cualquier cosa que no sea un dígito

Clases especiales de personajes

No Señor.	Ejemplo y descripción
1	<code>.</code> Coincide con cualquier personaje excepto la nueva línea
2	<code>\re</code> Hacer coincidir un dígito: [0-9]
3	<code>\RE</code> Coincidir con un no dígito: [^ 0-9]
4 4	<code>\s</code> Haga coincidir un carácter de espacio en blanco: [\ t \ r \ n \ f]
5 5	<code>\S</code> Coincidencia de espacios no en blanco: [^\ t \ r \ n \ f]
6 6	<code>\w</code> Haga coincidir un carácter de una sola palabra: [A-Za-z0-9_]
7 7	<code>\W</code> Haga coincidir un carácter que no sea una palabra: [^ A-Za-z0-9_]

Casos de repetición

No Señor.	Ejemplo y descripción
1	<code>¿rubí?</code> Haga coincidir "rub" o "ruby": la y es opcional
2	<code>rubí*</code>

	Coincide con "frotar" más 0 o más ys
3	rubí + Empareja "frotar" más 1 o más ys
4 4	\d {3} Iguala exactamente 3 dígitos
5 5	\d {3,} Match 3 o más dígitos
6 6	\d {3,5} Match 3, 4 o 5 dígitos

Repetición no cruel

Esto coincide con el menor número de repeticiones:

No Señor.	Ejemplo y descripción
1	<. *> Repetición codiciosa: coincide con "<python> perl>"
2	<. *?> No codicioso: coincide con "<python>" en "<python> perl>"

Agrupación con paréntesis

No Señor.	Ejemplo y descripción
1	\D\d + Sin grupo: + repeticiones \ d
2	(\D\d) +

	Agrupado: + repeticiones \ D \ d par
3	([Pp] ython (,)?) + Haga coincidir "Python", "Python, python, python", etc.

Referencias

Esto coincide con un grupo previamente emparejado nuevamente:

No Señor.	Ejemplo y descripción
1	([Pp]) ython & \ 1ails Coincide con python y cubos o Python y cubos
2	(["']) [^\ 1] * \ 1 Cadena de comillas simples o dobles. \ 1 coincide con lo que coincida con el primer grupo. \ 2 coincide con lo que coincida con el segundo grupo, etc.

Alternativas

No Señor.	Ejemplo y descripción
1	python perl Haga coincidir "python" o "perl"
2	frotar (y le) Empareja "rubí" o "rublo"
3	Python (! + \?) "Python" seguido de uno o más! o uno ?

Anclas

Esto necesita especificar la posición del partido.

No Señor.	Ejemplo y descripción
1	^ Python Haga coincidir "Python" al comienzo de una cadena o línea interna
2	Python \$ Haga coincidir "Python" al final de una cadena o línea
3	\ APython Haga coincidir "Python" al comienzo de una cadena
4 4	Python \ Z Haga coincidir "Python" al final de una cadena
5 5	\ bPython \ b Haga coincidir "Python" en un límite de palabra
6 6	\ brub \ B \ B es el límite de una palabra: coincide con "rub" en "rube" y "ruby" pero no solo
7 7	Python (? =!) Haga coincidir "Python", si va seguido de un signo de exclamación.
8	Pitón(?!!) Haga coincidir "Python", si no seguido de un signo de exclamación.

Sintaxis especial con paréntesis

No Señor.	Ejemplo y descripción
1	R (? # Comentario) Partidos "R". Todo lo demás es un comentario.

2	R (? l) uby No distingue entre mayúsculas y minúsculas mientras coincide con "uby"
3	R (? l: uby) Lo mismo que arriba
4 4	frotar (? : y le) Agrupar solo sin crear \ 1 referencia

Python - Programación CGI

La Common Gateway Interface, o CGI, es un conjunto de estándares que definen cómo se intercambia la información entre el servidor web y un script personalizado. Las especificaciones CGI son mantenidas actualmente por la NCSA.

¿Qué es el CGI?

- La interfaz de puerta de enlace común, o CGI, es un estándar para que los programas de puerta de enlace externos interactúen con servidores de información como los servidores HTTP.
- La versión actual es CGI / 1.1 y CGI / 1.2 está en progreso.

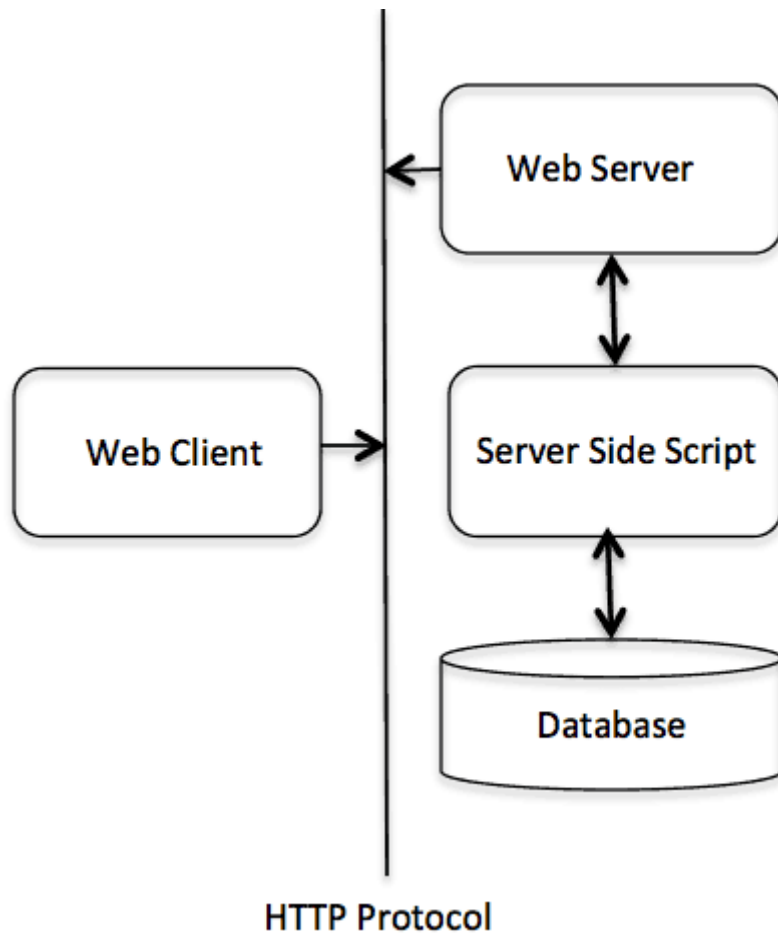
Buscando en la web

Para comprender el concepto de CGI, veamos qué sucede cuando hacemos clic en un hipervínculo para navegar por una página web o URL en particular.

- Su navegador se pone en contacto con el servidor web HTTP y exige la URL, es decir, el nombre del archivo.
- El servidor web analiza la URL y busca el nombre de archivo. Si encuentra ese archivo, lo devuelve al navegador, de lo contrario, envía un mensaje de error que indica que solicitó un archivo incorrecto.
- El navegador web toma la respuesta del servidor web y muestra el archivo recibido o el mensaje de error.

Sin embargo, es posible configurar el servidor HTTP para que cada vez que se solicite un archivo en un determinado directorio no se envíe de vuelta; en su lugar, se ejecuta como un programa, y lo que ese programa genera se devuelve para que su navegador lo muestre. Esta función se llama Common Gateway Interface o CGI y los programas se llaman scripts CGI. Estos programas CGI pueden ser Python Script, PERL Script, Shell Script, programa C o C ++, etc.

Diagrama de arquitectura CGI



Soporte y configuración del servidor web

Antes de continuar con la Programación CGI, asegúrese de que su Servidor Web sea compatible con CGI y que esté configurado para manejar Programas CGI. Todos los programas CGI que debe ejecutar el servidor HTTP se guardan en un directorio preconfigurado. Este directorio se llama Directorio CGI y, por convención, se denomina `/var/www/cgi-bin`. Por convención, los archivos CGI tienen extensión como **.cgi**, pero también puede mantener sus archivos con la extensión **.py** de python .

De manera predeterminada, el servidor Linux está configurado para ejecutar solo los scripts en el directorio `cgi-bin` en `/var/www`. Si desea especificar cualquier otro directorio para ejecutar sus scripts CGI, comente las siguientes líneas en el archivo `httpd.conf`:

```
<Directory "/var/www/cgi-bin">  
    AllowOverride None  
    Options ExecCGI  
    Order allow,deny  
    Allow from all  
</Directory>
```

```
<Directory "/var/www/cgi-bin">
Options All
</Directory>
```

Aquí, suponemos que tiene un servidor web funcionando correctamente y que puede ejecutar cualquier otro programa CGI como Perl o Shell, etc.

Primer programa CGI

Aquí hay un enlace simple, que está vinculado a un script CGI llamado [hello.py](#). Este archivo se mantiene en el directorio / var / www / cgi-bin y tiene el siguiente contenido. Antes de ejecutar su programa CGI, asegúrese de haber cambiado el modo de archivo usando el comando **chmod 755 hello.py** UNIX para hacer que el archivo sea ejecutable.

```
#!/usr/bin/python

print "Content-type:text/html\r\n\r\n"
print '<html>'
print '<head>'
print '<title>Hello Word - First CGI Program</title>'
print '</head>'
print '<body>'
print '<h2>Hello Word! This is my first CGI program</h2>'
print '</body>'
print '</html>'
```

Si hace clic en [hello.py](#), esto produce el siguiente resultado:

Hola palabra Este es mi primer programa CGI

Este script [hello.py](#) es un script simple de Python, que escribe su salida en el archivo STDOUT, es decir, en la pantalla. Hay una característica importante y adicional disponible que es la primera línea que se imprimirá. **Tipo de contenido: texto / html \ r \ n \ r \ n**. Esta línea se envía de vuelta al navegador y especifica el tipo de contenido que se mostrará en la pantalla del navegador.

A estas alturas ya debe haber entendido el concepto básico de CGI y puede escribir muchos programas CGI complicados utilizando Python. Este script puede interactuar con cualquier otro sistema externo también para intercambiar información como RDBMS.

Encabezado HTTP

La línea **Content-type: text / html \ r \ n \ r \ n** forma parte del encabezado HTTP que se envía al navegador para comprender el contenido. Todo el encabezado HTTP estará en la siguiente forma:

```
HTTP Field Name: Field Content
```

For Example

Content-type: text/html\r\n\r\n

Hay algunos otros encabezados HTTP importantes, que utilizará con frecuencia en su programación CGI.

No Señor.	Encabezado y descripción
1	Tipo de contenido: Una cadena MIME que define el formato del archivo que se devuelve. El ejemplo es Content-type: text / html
2	Caduca: fecha La fecha en que la información deja de ser válida. El navegador lo utiliza para decidir cuándo se debe actualizar una página. Una cadena de fecha válida tiene el formato 01 de enero de 1998 12:00:00 GMT.
3	Ubicación: URL La URL que se devuelve en lugar de la URL solicitada. Puede usar este campo para redirigir una solicitud a cualquier archivo.
4 4	Última modificación: fecha La fecha de la última modificación del recurso.
5 5	Longitud del contenido: N La longitud, en bytes, de los datos que se devuelven. El navegador utiliza este valor para informar el tiempo estimado de descarga de un archivo.
6 6	Set-Cookie: Cadena Establecer la cookie pasada a través de la <i>cadena</i>

Variables de entorno CGI

Todos los programas CGI tienen acceso a las siguientes variables de entorno. Estas variables juegan un papel importante al escribir cualquier programa CGI.

No Señor.	Nombre y descripción de la variable
1	TIPO DE CONTENIDO

	<p>El tipo de datos del contenido. Se usa cuando el cliente envía contenido adjunto al servidor. Por ejemplo, carga de archivos.</p>
2	<p>LARGANCIA DE CONTENIDO</p> <p>La longitud de la información de la consulta. Está disponible solo para solicitudes POST.</p>
3	<p>HTTP_COOKIE</p> <p>Devuelve las cookies establecidas en forma de par clave y valor.</p>
4 4	<p>HTTP_USER_AGENT</p> <p>El campo de encabezado de solicitud de agente de usuario contiene información sobre el agente de usuario que origina la solicitud. Es el nombre del navegador web.</p>
5 5	<p>RUTA_INFO</p> <p>El camino para el script CGI.</p>
6 6	<p>QUERY_STRING</p> <p>La información codificada en URL que se envía con la solicitud del método GET.</p>
7 7	<p>REMOTE_ADDR</p> <p>La dirección IP del host remoto que realiza la solicitud. Este es un registro útil o para autenticación.</p>
8	<p>SERVIDOR REMOTO</p> <p>El nombre completo del host que realiza la solicitud. Si esta información no está disponible, entonces REMOTE_ADDR se puede usar para obtener la dirección IR.</p>
9 9	<p>SOLICITUD_MÉTODO</p> <p>El método utilizado para realizar la solicitud. Los métodos más comunes son GET y POST.</p>
10	<p>SCRIPT_FILENAME</p> <p>La ruta completa al script CGI.</p>
11	<p>SCRIPT_NAME</p> <p>El nombre del script CGI.</p>

12	<p>NOMBRE DEL SERVIDOR</p> <p>El nombre de host del servidor o la dirección IP</p>
13	<p>SERVER_SOFTWARE</p> <p>El nombre y la versión del software que ejecuta el servidor.</p>

Aquí hay un pequeño programa CGI para enumerar todas las variables CGI. Haga clic en este enlace para ver el resultado [Obtener entorno](#)

```
#!/usr/bin/python

import os

print "Content-type: text/html\r\n\r\n";
print "<font size=+1>Environment</font><\br>";
for param in os.environ.keys():
    print "<b>%20s</b>: %s<\br>" % (param, os.environ[param])
```

Métodos GET y POST

Debe haber encontrado muchas situaciones en las que necesita pasar información de su navegador al servidor web y, en última instancia, a su Programa CGI. Con mayor frecuencia, el navegador utiliza dos métodos: dos pasan esta información al servidor web. Estos métodos son el método GET y el método POST.

Pasando información usando el método GET

El método GET envía la información codificada del usuario adjunta a la solicitud de página. La página y la información codificada están separadas por el? personaje de la siguiente manera:

`http://www.test.com/cgi-bin/hello.py?key1=value1&key2=value2`

El método GET es el método predeterminado para pasar información del navegador al servidor web y produce una cadena larga que aparece en el cuadro Ubicación del navegador. Nunca use el método GET si tiene una contraseña u otra información confidencial para pasar al servidor. El método GET tiene una limitación de tamaño: solo se pueden enviar 1024 caracteres en una cadena de solicitud. El método GET envía información utilizando el encabezado QUERY_STRING y estará accesible en su programa CGI a través de la variable de entorno QUERY_STRING.

Puede pasar información simplemente concatenando pares clave y valor junto con cualquier URL o puede usar etiquetas HTML <FORM> para pasar información usando el método GET.

Ejemplo de URL simple: método de obtención

Aquí hay una URL simple, que pasa dos valores al programa hello_get.py usando el método GET.

/cgi-bin/hello_get.py?first_name=ZARA&last_name=ALI

A continuación se **muestra el script hello_get.py** para manejar la entrada dada por el navegador web. Vamos a utilizar el módulo **cgi**, lo que hace que sea muy fácil acceder a la información que se pasa.

```
#!/usr/bin/python

# Import modules for CGI handling
import cgi, cgitb

# Create instance of FieldStorage
form = cgi.FieldStorage()

# Get data from fields
first_name = form.getvalue('first_name')
last_name = form.getvalue('last_name')

print "Content-type:text/html\r\n\r\n"
print "<html>"
print "<head>"
print "<title>Hello - Second CGI Program</title>"
print "</head>"
print "<body>"
print "<h2>Hello %s %s</h2>" % (first_name, last_name)
print "</body>"
print "</html>"
```

Esto generaría el siguiente resultado:

Hola zara ali

Ejemplo de FORMA Simple: Método GET

Este ejemplo pasa dos valores usando HTML FORM y el botón de enviar. Utilizamos el mismo script CGI hello_get.py para manejar esta entrada.

```
<form action = "/cgi-bin/hello_get.py" method = "get">
First Name: <input type = "text" name = "first_name"> <br />

Last Name: <input type = "text" name = "last_name" />
<input type = "submit" value = "Submit" />
</form>
```

Aquí está el resultado real del formulario anterior, ingrese Nombre y Apellido y luego haga clic en el botón Enviar para ver el resultado.

Nombre de pila:

Apellido:

Pasando información usando el método POST

Un método generalmente más confiable de pasar información a un programa CGI es el método POST. Esto empaqueta la información exactamente de la misma manera que los métodos GET, pero en lugar de enviarla como una cadena de texto después de un? en la URL lo envía como un mensaje separado. Este mensaje entra en el script CGI en forma de entrada estándar.

A continuación se muestra el mismo script `hello_get.py` que maneja el método GET y POST.

```
#!/usr/bin/python

# Import modules for CGI handling
import cgi, cgitb

# Create instance of FieldStorage
form = cgi.FieldStorage()

# Get data from fields
first_name = form.getvalue('first_name')
last_name = form.getvalue('last_name')

print "Content-type:text/html\r\n\r\n"
print "<html>"
print "<head>"
print "<title>Hello - Second CGI Program</title>"
print "</head>"
print "<body>"
print "<h2>Hello %s %s</h2>" % (first_name, last_name)
print "</body>"
print "</html>"
```

Tomemos nuevamente el mismo ejemplo anterior que pasa dos valores usando HTML FORM y el botón de enviar. Utilizamos el mismo script CGI `hello_get.py` para manejar esta entrada.

```
<form action = "/cgi-bin/hello_get.py" method = "post">
First Name: <input type = "text" name = "first_name"><br />
Last Name: <input type = "text" name = "last_name" />

<input type = "submit" value = "Submit" />
</form>
```

Aquí está la salida real de la forma anterior. Ingrese Nombre y Apellido y luego haga clic en el botón Enviar para ver el resultado.

Nombre de pila:

Apellido:

Pasar los datos de la casilla de verificación al programa CGI

Las casillas de verificación se usan cuando se requiere seleccionar más de una opción.

Aquí hay un código HTML de ejemplo para un formulario con dos casillas de verificación:

```
<form action = "/cgi-bin/checkbox.cgi" method = "POST" target = "_blank">
<input type = "checkbox" name = "maths" value = "on" /> Maths
<input type = "checkbox" name = "physics" value = "on" />
Physics
<input type = "submit" value = "Select Subject" />
</form>
```

El resultado de este código es la siguiente forma:

Matemáticas Física

A continuación se muestra el script checkbox.cgi para manejar la entrada dada por el navegador web para el botón de casilla de verificación.

```
#!/usr/bin/python

# Import modules for CGI handling
import cgi, cgitb

# Create instance of FieldStorage
form = cgi.FieldStorage()

# Get data from fields
if form.getvalue('maths'):
    math_flag = "ON"
else:
    math_flag = "OFF"

if form.getvalue('physics'):
    physics_flag = "ON"
else:
    physics_flag = "OFF"

print "Content-type:text/html\r\n\r\n"
print "<html>"
print "<head>"
print "<title>Checkbox - Third CGI Program</title>"
print "</head>"
print "<body>"
print "<h2> CheckBox Maths is : %s</h2>" % math_flag
print "<h2> CheckBox Physics is : %s</h2>" % physics_flag
print "</body>"
print "</html>"
```

Pasar datos de botón de radio al programa CGI

Los botones de radio se usan cuando solo se requiere seleccionar una opción.

Aquí hay un código HTML de ejemplo para un formulario con dos botones de opción:

```
<form action = "/cgi-bin/radiobutton.py" method = "post"
target = "_blank">
<input type = "radio" name = "subject" value = "maths" />
Maths
<input type = "radio" name = "subject" value = "physics" />
Physics
<input type = "submit" value = "Select Subject" />
</form>
```

El resultado de este código es la siguiente forma:

Matemáticas Física

A continuación se muestra el script radiobutton.py para manejar la entrada dada por el navegador web para el botón de opción:

```
#!/usr/bin/python

# Import modules for CGI handling
import cgi, cgitb

# Create instance of FieldStorage
form = cgi.FieldStorage()

# Get data from fields
if form.getvalue('subject'):
    subject = form.getvalue('subject')
else:
    subject = "Not set"

print "Content-type:text/html\r\n\r\n"
print "<html>"
print "<head>"
print "<title>Radio - Fourth CGI Program</title>"
print "</head>"
print "<body>"
print "<h2> Selected Subject is %s</h2>" % subject
print "</body>"
print "</html>"
```

Pasar datos del área de texto al programa CGI

El elemento TEXTAREA se usa cuando el texto multilínea tiene que pasar al Programa CGI.

Aquí hay un código HTML de ejemplo para un formulario con un cuadro TEXTAREA:

```
<form action = "/cgi-bin/textarea.py" method = "post" target = "_blank">
<textarea name = "textcontent" cols = "40" rows = "4">
Type your text here...
</textarea>
<input type = "submit" value = "Submit" />
</form>
```

El resultado de este código es la siguiente forma:

A screenshot of a web browser window. The browser's address bar is empty. The main content area shows a text input field with the placeholder text "Type your text here...". To the right of the text field is a button labeled "Enviar". The browser's scrollbar is visible on the right side.

A continuación se muestra el script textarea.cgi para manejar la entrada dada por el navegador web:

```
#!/usr/bin/python

# Import modules for CGI handling
import cgi, cgitb

# Create instance of FieldStorage
form = cgi.FieldStorage()

# Get data from fields
if form.getvalue('textcontent'):
    text_content = form.getvalue('textcontent')
else:
    text_content = "Not entered"

print "Content-type:text/html\r\n\r\n"
print "<html>"
print "<head>";
print "<title>Text Area - Fifth CGI Program</title>"
print "</head>"
print "<body>"
print "<h2> Entered Text Content is %s</h2>" % text_content
print "</body>"
```

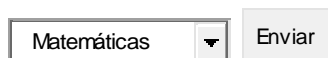
Pasar datos de cuadro desplegable al programa CGI

El cuadro desplegable se utiliza cuando tenemos muchas opciones disponibles, pero solo se seleccionarán una o dos.

Aquí hay un código HTML de ejemplo para un formulario con un cuadro desplegable:

```
<form action = "/cgi-bin/dropdown.py" method = "post" target = "_blank">
<select name = "dropdown">
<option value = "Maths" selected>Maths</option>
<option value = "Physics">Physics</option>
</select>
<input type = "submit" value = "Submit"/>
</form>
```

El resultado de este código es la siguiente forma:



A screenshot of a web form. It features a dropdown menu with the text 'Matemáticas' and a small downward arrow. To the right of the dropdown is a button labeled 'Enviar'.

A continuación se muestra el script dropdown.py para manejar la entrada dada por el navegador web.

```
#!/usr/bin/python

# Import modules for CGI handling
import cgi, cgitb

# Create instance of FieldStorage
form = cgi.FieldStorage()

# Get data from fields
if form.getvalue('dropdown'):
    subject = form.getvalue('dropdown')
else:
    subject = "Not entered"

print "Content-type:text/html\r\n\r\n"
print "<html>"
print "<head>"
print "<title>Dropdown Box - Sixth CGI Program</title>"
print "</head>"
print "<body>"
print "<h2> Selected Subject is %s</h2>" % subject
print "</body>"
print "</html>"
```

Uso de cookies en CGI

El protocolo HTTP es un protocolo sin estado. Para un sitio web comercial, se requiere mantener la información de la sesión entre diferentes páginas. Por ejemplo, el registro de un usuario finaliza después de completar muchas páginas. ¿Cómo mantener la información de la sesión del usuario en todas las páginas web?

En muchas situaciones, el uso de cookies es el método más eficiente para recordar y rastrear preferencias, compras, comisiones y otra información requerida para una mejor experiencia del visitante o estadísticas del sitio.

¿Cómo funciona?

Su servidor envía algunos datos al navegador del visitante en forma de cookie. El navegador puede aceptar la cookie. Si lo hace, se almacena como un registro de texto sin formato en el disco duro del visitante. Ahora, cuando el visitante llega a otra página de su sitio, la cookie está disponible para su recuperación. Una vez recuperado, su servidor sabe / recuerda lo que estaba almacenado.

Las cookies son un registro de datos de texto sin formato de 5 campos de longitud variable:

- **Caduca** : la fecha de caducidad de la cookie. Si está en blanco, la cookie caducará cuando el visitante cierre el navegador.
- **Dominio** : el nombre de dominio de su sitio.
- **Ruta** : la ruta al directorio o página web que establece la cookie. Esto puede estar en blanco si desea recuperar la cookie de cualquier directorio o página.
- **Seguro** : si este campo contiene la palabra "seguro", la cookie solo se puede recuperar con un servidor seguro. Si este campo está en blanco, no existe tal restricción.
- **Nombre = Valor** : las cookies se configuran y recuperan en forma de pares de clave y valor.

Configurar cookies

Es muy fácil enviar cookies al navegador. Estas cookies se envían junto con el Encabezado HTTP antes al campo Tipo de contenido. Suponiendo que desea establecer la identificación de usuario y la contraseña como cookies. La configuración de las cookies se realiza de la siguiente manera:

```
#!/usr/bin/python

print "Set-Cookie:UserID = XYZ;\r\n"
print "Set-Cookie:Password = XYZ123;\r\n"
print "Set-Cookie:Expires = Tuesday, 31-Dec-2007 23:12:40
GMT";\r\n"
print "Set-Cookie:Domain = www.tutorialspoint.com;\r\n"
print "Set-Cookie:Path = /perl;\n"
print "Content-type:text/html\r\n\r\n"
.....Rest of the HTML Content.....
```

A partir de este ejemplo, debe haber entendido cómo configurar las cookies. Utilizamos el encabezado HTTP **Set-Cookie** para establecer cookies.

Es opcional establecer atributos de cookies como Caduca, Dominio y Ruta. Es notable que las cookies se establezcan antes de enviar la línea mágica "**Tipo de contenido: texto / html \r \n \r \n**".

Recuperando Cookies

Es muy fácil recuperar todas las cookies establecidas. Las cookies se almacenan en la variable de entorno CGI HTTP_COOKIE y tendrán la siguiente forma:

```
key1 = value1;key2 = value2;key3 = value3....
```

Aquí hay un ejemplo de cómo recuperar cookies.

```
#!/usr/bin/python

# Import modules for CGI handling
from os import environ
import cgi, cgitb

if environ.has_key('HTTP_COOKIE'):
    for cookie in map(strip, split(environ['HTTP_COOKIE'],
';')):
        (key, value ) = split(cookie, '=');
        if key == "UserID":
            user_id = value

        if key == "Password":
            password = value

print "User ID = %s" % user_id
print "Password = %s" % password
```

Esto produce el siguiente resultado para las cookies establecidas por el script anterior:

```
User ID = XYZ
Password = XYZ123
```

Ejemplo de carga de archivos

Para cargar un archivo, el formulario HTML debe tener el atributo enctype establecido en **multipart / form-data** . La etiqueta de entrada con el tipo de archivo crea un botón "Examinar".

```
<html>
<body>
  <form enctype = "multipart/form-data"
        action = "save_file.py" method = "post">
    <p>File: <input type = "file" name = "filename" /></p>
    <p><input type = "submit" value = "Upload" /></p>
  </form>
</body>
</html>
```

El resultado de este código es la siguiente forma:

Archivo:

El ejemplo anterior se ha deshabilitado intencionalmente para salvar a las personas que cargan archivos en nuestro servidor, pero puede probar el código anterior con su servidor.

Aquí está el script **save_file.py** para manejar la carga de archivos:

```
#!/usr/bin/python

import cgi, os
import cgi; cgi.enable()

form = cgi.FieldStorage()

# Get filename here.
fileitem = form['filename']

# Test if the file was uploaded
if fileitem.filename:
    # strip leading path from file name to avoid
    # directory traversal attacks
    fn = os.path.basename(fileitem.filename)
    open('/tmp/' + fn, 'wb').write(fileitem.file.read())

    message = 'The file "' + fn + '" was uploaded
successfully'

else:
    message = 'No file was uploaded'

print """\
Content-Type: text/html\n
<html>
<body>
    <p>%s</p>
</body>
</html>
""" % (message,)
```

Si ejecuta el script anterior en Unix / Linux, debe ocuparse de reemplazar el separador de archivos de la siguiente manera; de lo contrario, en su máquina de Windows, la instrucción `open ()` debería funcionar bien.

```
fn = os.path.basename(fileitem.filename.replace("\\", "/" ))
```

¿Cómo generar un cuadro de diálogo "Descarga de archivos"?

A veces, se desea que desee dar una opción donde un usuario puede hacer clic en un enlace y aparecerá un cuadro de diálogo "Descarga de archivos" para el usuario en lugar de mostrar contenido real. Esto es muy fácil y se puede lograr a través del encabezado HTTP. Este encabezado HTTP es diferente del encabezado mencionado en la sección anterior.

Por ejemplo, si desea que un archivo **FileName** se pueda descargar desde un enlace determinado, su sintaxis es la siguiente:

```
#!/usr/bin/python

# HTTP Header
print "Content-Type:application/octet-stream; name =
\"FileName\"\\r\\n";
print "Content-Disposition: attachment; filename =
\"FileName\"\\r\\n\\n";

# Actual File Content will go here.
fo = open("foo.txt", "rb")

str = fo.read();
print str

# Close opened file
fo.close()
```

Espero que hayan disfrutado este tutorial. En caso afirmativo, envíeme sus comentarios a: [Contáctenos](#)

Python - Acceso a la base de datos MySQL

El estándar de Python para las interfaces de bases de datos es la API de Python DB. La mayoría de las interfaces de bases de datos Python se adhieren a este estándar.

Puede elegir la base de datos adecuada para su aplicación. Python Database API admite una amplia gama de servidores de bases de datos, como:

- GadFly

- mSQL
- MySQL
- PostgreSQL
- Microsoft SQL Server 2000
- Informix
- Interbase
- Oracle
- Sybase
-

Aquí está la lista de interfaces de base de datos Python disponibles: [Interfaces de base de datos Python y API](#) . Debe descargar un módulo API DB diferente para cada base de datos a la que necesite acceder. Por ejemplo, si necesita acceder a una base de datos Oracle y a una base de datos MySQL, debe descargar los módulos de base de datos Oracle y MySQL.

La API DB proporciona un estándar mínimo para trabajar con bases de datos utilizando estructuras y sintaxis de Python siempre que sea posible. Esta API incluye lo siguiente:

- Importando el módulo API.
- Adquirir una conexión con la base de datos.
- Emisión de sentencias SQL y procedimientos almacenados.
- Cerrando la conexión

Aprenderíamos todos los conceptos usando MySQL, así que hablemos sobre el módulo MySQLdb.

¿Qué es MySQLdb?

MySQLdb es una interfaz para conectarse a un servidor de base de datos MySQL desde Python. Implementa la Python Database API v2.0 y está construida sobre la API MySQL C.

¿Cómo instalo MySQLdb?

Antes de continuar, asegúrese de tener MySQLdb instalado en su máquina. Simplemente escriba lo siguiente en su script de Python y ejecútelo:

```
#!/usr/bin/python

import MySQLdb
```

Si produce el siguiente resultado, significa que el módulo MySQLdb no está instalado:

```
Traceback (most recent call last):
  File "test.py", line 3, in <module>
    import MySQLdb
ImportError: No module named MySQLdb
```

Para instalar el módulo MySQLdb, use el siguiente comando:

```
For Ubuntu, use the following command -  
$ sudo apt-get install python-pip python-dev libmysqlclient-dev
```

```
For Fedora, use the following command -  
$ sudo dnf install python python-devel mysql-devel redhat-rpm-config gcc
```

```
For Python command prompt, use the following command -  
pip install MySQL-python
```

Nota : asegúrese de tener privilegios de root para instalar el módulo anterior.

Conexión de base de datos

Antes de conectarse a una base de datos MySQL, asegúrese de lo siguiente:

- Ha creado una base de datos TESTDB.
- Ha creado una tabla EMPLEADO en TESTDB.
- Esta tabla tiene los campos FIRST_NAME, LAST_NAME, AGE, SEX e INCOME.
- El ID de usuario "testuser" y la contraseña "test123" están configurados para acceder a TESTDB.
- El módulo Python MySQLdb está instalado correctamente en su máquina.
- Has seguido el tutorial de MySQL para comprender los conceptos básicos de MySQL.

Ejemplo

El siguiente es el ejemplo de conexión con la base de datos MySQL "TESTDB"

```
#!/usr/bin/python  
  
import MySQLdb  
  
# Open database connection  
db =  
MySQLdb.connect("localhost","testuser","test123","TESTDB" )  
  
# prepare a cursor object using cursor() method  
cursor = db.cursor()  
  
# execute SQL query using execute() method.  
cursor.execute("SELECT VERSION()")  
  
# Fetch a single row using fetchone() method.  
data = cursor.fetchone()  
print "Database version : %s " % data  
  
# disconnect from server  
db.close()
```

Mientras ejecuta este script, produce el siguiente resultado en mi máquina Linux.

```
Database version : 5.0.45
```

Si se establece una conexión con el origen de datos, se devuelve un Objeto de conexión y se guarda en **db** para su uso posterior; de lo contrario, **db** se establece en Ninguno. A continuación, el objeto **db** se usa para crear un objeto de **cursor**, que a su vez se usa para ejecutar consultas SQL. Finalmente, antes de salir, asegura que la conexión de la base de datos esté cerrada y que se liberen los recursos.

Crear tabla de base de datos

Una vez que se establece una conexión a la base de datos, estamos listos para crear tablas o registros en las tablas de la base de datos utilizando el método de **ejecución** del cursor creado.

Ejemplo

Vamos a crear la tabla de base de datos EMPLEADO -

```
#!/usr/bin/python

import MySQLdb

# Open database connection
db =
MySQLdb.connect("localhost","testuser","test123","TESTDB" )

# prepare a cursor object using cursor() method
cursor = db.cursor()

# Drop table if it already exist using execute() method.
cursor.execute("DROP TABLE IF EXISTS EMPLOYEE")

# Create table as per requirement
sql = """CREATE TABLE EMPLOYEE (
        FIRST_NAME  CHAR(20) NOT NULL,
        LAST_NAME   CHAR(20),
        AGE         INT,
        SEX         CHAR(1),
        INCOME      FLOAT )"""

cursor.execute(sql)

# disconnect from server
db.close()
```

INSERTAR Operación

Es obligatorio cuando desea crear sus registros en una tabla de base de datos.

Ejemplo

El siguiente ejemplo, ejecuta la *instrucción* SQL *INSERT* para crear un registro en la tabla EMPLOYEE:

```
#!/usr/bin/python

import MySQLdb

# Open database connection
db =
MySQLdb.connect("localhost","testuser","test123","TESTDB" )

# prepare a cursor object using cursor() method
cursor = db.cursor()

# Prepare SQL query to INSERT a record into the database.
sql = """INSERT INTO EMPLOYEE(FIRST_NAME,
          LAST_NAME, AGE, SEX, INCOME)
          VALUES ('Mac', 'Mohan', 20, 'M', 2000)"""

try:
    # Execute the SQL command
    cursor.execute(sql)
    # Commit your changes in the database
    db.commit()
except:
    # Rollback in case there is any error
    db.rollback()

# disconnect from server
db.close()
```

El ejemplo anterior se puede escribir de la siguiente manera para crear consultas SQL dinámicamente:

```
#!/usr/bin/python

import MySQLdb

# Open database connection
db =
MySQLdb.connect("localhost","testuser","test123","TESTDB" )

# prepare a cursor object using cursor() method
cursor = db.cursor()

# Prepare SQL query to INSERT a record into the database.
sql = "INSERT INTO EMPLOYEE(FIRST_NAME, \
          LAST_NAME, AGE, SEX, INCOME) \
          VALUES ('%s', '%s', '%d', '%c', '%d' )" % \
```

```

        ('Mac', 'Mohan', 20, 'M', 2000)
try:
    # Execute the SQL command
    cursor.execute(sql)
    # Commit your changes in the database
    db.commit()
except:
    # Rollback in case there is any error
    db.rollback()

# disconnect from server
db.close()

```

Ejemplo

El siguiente segmento de código es otra forma de ejecución donde puede pasar parámetros directamente -

```

.....
user_id = "test123"
password = "password"

con.execute('insert into Login values("%s", "%s")' % \
            (user_id, password))
.....

```

Operación LEER

La operación READ en cualquier base de datos significa obtener información útil de la base de datos.

Una vez que se establece nuestra conexión de base de datos, está listo para realizar una consulta en esta base de datos. Puede usar el método **fetchone ()** para obtener un registro único o el método **fetchall ()** para fechizar múltiples valores de una tabla de base de datos.

- **fetchone ()** : recupera la siguiente fila de un conjunto de resultados de la consulta. Un conjunto de resultados es un objeto que se devuelve cuando se usa un objeto cursor para consultar una tabla.
- **fetchall ()** : recupera todas las filas de un conjunto de resultados. Si algunas filas ya se han extraído del conjunto de resultados, recupera las filas restantes del conjunto de resultados.
- **rowcount** : este es un atributo de solo lectura y devuelve el número de filas afectadas por un método execute ().

Ejemplo

El siguiente procedimiento consulta todos los registros de la tabla EMPLEADO con un salario superior a 1000:

```
#!/usr/bin/python
```

```

import MySQLdb

# Open database connection
db =
MySQLdb.connect("localhost","testuser","test123","TESTDB" )

# prepare a cursor object using cursor() method
cursor = db.cursor()

sql = "SELECT * FROM EMPLOYEE \
      WHERE INCOME > '%d'" % (1000)
try:
    # Execute the SQL command
    cursor.execute(sql)
    # Fetch all the rows in a list of lists.
    results = cursor.fetchall()
    for row in results:
        fname = row[0]
        lname = row[1]
        age = row[2]
        sex = row[3]
        income = row[4]
        # Now print fetched result
        print "fname=%s,lname=%s,age=%d,sex=%s,income=%d" % \
              (fname, lname, age, sex, income )
except:
    print "Error: unable to fetch data"

# disconnect from server
db.close()

```

Esto producirá el siguiente resultado:

```
fname=Mac, lname=Mohan, age=20, sex=M, income=2000
```

Operación de actualización

ACTUALIZAR La operación en cualquier base de datos significa actualizar uno o más registros, que ya están disponibles en la base de datos.

El siguiente procedimiento actualiza todos los registros que tienen SEXO como 'M'. Aquí, aumentamos la EDAD de todos los hombres en un año.

Ejemplo

```

#!/usr/bin/python

import MySQLdb

# Open database connection
db =
MySQLdb.connect("localhost","testuser","test123","TESTDB" )

```

```

# prepare a cursor object using cursor() method
cursor = db.cursor()

# Prepare SQL query to UPDATE required records
sql = "UPDATE EMPLOYEE SET AGE = AGE + 1
      WHERE SEX = '%c'" % ('M')

try:
    # Execute the SQL command
    cursor.execute(sql)
    # Commit your changes in the database
    db.commit()
except:
    # Rollback in case there is any error
    db.rollback()

# disconnect from server
db.close()

```

Operación ELIMINAR

La operación ELIMINAR es necesaria cuando desea eliminar algunos registros de su base de datos. El siguiente es el procedimiento para eliminar todos los registros del EMPLEADO donde la EDAD es más de 20 -

Ejemplo

```

#!/usr/bin/python

import MySQLdb

# Open database connection
db =
MySQLdb.connect("localhost","testuser","test123","TESTDB" )

# prepare a cursor object using cursor() method
cursor = db.cursor()

# Prepare SQL query to DELETE required records
sql = "DELETE FROM EMPLOYEE WHERE AGE > '%d'" % (20)
try:
    # Execute the SQL command
    cursor.execute(sql)
    # Commit your changes in the database
    db.commit()
except:
    # Rollback in case there is any error
    db.rollback()

# disconnect from server
db.close()

```


Realizar transacciones

Las transacciones son un mecanismo que garantiza la coherencia de los datos. Las transacciones tienen las siguientes cuatro propiedades:

- **Atomicidad** : se completa una transacción o no sucede nada.
- **Consistencia** : una transacción debe comenzar en un estado consistente y dejar el sistema en un estado consistente.
- **Aislamiento** : los resultados intermedios de una transacción no son visibles fuera de la transacción actual.
- **Durabilidad** : una vez que se **confirmó** una transacción, los efectos son persistentes, incluso después de una falla del sistema.

Python DB API 2.0 proporciona dos métodos para *confirmar* o *revertir* una transacción.

Ejemplo

Ya sabes cómo implementar transacciones. Aquí hay de nuevo un ejemplo similar:

```
# Prepare SQL query to DELETE required records
sql = "DELETE FROM EMPLOYEE WHERE AGE > '%d'" % (20)
try:
    # Execute the SQL command
    cursor.execute(sql)
    # Commit your changes in the database
    db.commit()
except:
    # Rollback in case there is any error
    db.rollback()
```

Operación COMMIT

Commit es la operación, que da una señal verde a la base de datos para finalizar los cambios, y después de esta operación, no se puede revertir ningún cambio.

Aquí hay un ejemplo simple para llamar al método **commit** .

```
db.commit()
```

Operación ROLLBACK

Si no está satisfecho con uno o más de los cambios y desea revertirlos por completo, utilice el método **rollback ()** .

Aquí hay un ejemplo simple para llamar al método **rollback ()** .

```
db.rollback()
```

Desconectando la base de datos

Para desconectar la conexión de la base de datos, use el método `close()`.

```
db.close()
```

Si el usuario cierra la conexión a una base de datos con el método `close()`, la base de datos revierte cualquier transacción pendiente. Sin embargo, en lugar de depender de cualquiera de los detalles de implementación de nivel inferior de la base de datos, sería mejor que su aplicación llame explícitamente a `commit` o `rollback`.

Errores de manejo

Hay muchas fuentes de errores. Algunos ejemplos son un error de sintaxis en una instrucción SQL ejecutada, un error de conexión o la llamada al método `fetch` para un identificador de instrucción ya cancelado o terminado.

La API DB define una serie de errores que deben existir en cada módulo de base de datos. La siguiente tabla enumera estas excepciones.

No Señor.	Excepción y Descripción
1	Advertencia Utilizado para problemas no fatales. Debe subclase <code>StandardError</code> .
2	Error Clase base para errores. Debe subclase <code>StandardError</code> .
3	InterfaceError Se usa para errores en el módulo de base de datos, no en la base de datos en sí. Debe subclase <code>Error</code> .
4 4	Error de la base de datos Usado para errores en la base de datos. Debe subclase <code>Error</code> .
5 5	Error de datos Subclase de <code>DatabaseError</code> que se refiere a errores en los datos.
6 6	Error operacional Subclase de <code>DatabaseError</code> que se refiere a errores como la pérdida de una conexión a la base de datos. Estos errores generalmente están fuera del control del

	scripter de Python.
7 7	<p>IntegridadError</p> <p>Subclase de DatabaseError para situaciones que dañarían la integridad relacional, como restricciones de unicidad o claves foráneas.</p>
8	<p>Error interno</p> <p>Subclase de DatabaseError que se refiere a errores internos del módulo de base de datos, como un cursor que ya no está activo.</p>
9 9	<p>ProgrammingError</p> <p>Subclase de DatabaseError que se refiere a errores como un nombre de tabla incorrecto y otras cosas que se pueden culpar de forma segura.</p>
10	<p>NotSupportedError</p> <p>Subclase de DatabaseError que se refiere a intentar llamar a una funcionalidad no compatible.</p>

Sus scripts de Python deben manejar estos errores, pero antes de usar cualquiera de las excepciones anteriores, asegúrese de que su MySQLdb sea compatible con esa excepción. Puede obtener más información sobre ellos leyendo la especificación DB API 2.0.

Python - Programación en red

Python proporciona dos niveles de acceso a los servicios de red. A un nivel bajo, puede acceder al soporte básico de socket en el sistema operativo subyacente, lo que le permite implementar clientes y servidores para protocolos orientados y sin conexión.

Python también tiene bibliotecas que proporcionan acceso de nivel superior a protocolos de red específicos a nivel de aplicación, como FTP, HTTP, etc.

Este capítulo le brinda comprensión sobre el concepto más famoso en Redes: programación de sockets.

¿Qué son los sockets?

Los sockets son los puntos finales de un canal de comunicaciones bidireccionales. Los sockets pueden comunicarse dentro de un proceso, entre procesos en la misma máquina o entre procesos en diferentes continentes.

Los sockets se pueden implementar en varios tipos de canales diferentes: sockets de dominio Unix, TCP, UDP, etc. La biblioteca de *sockets* proporciona clases específicas para manejar los transportes comunes, así como una interfaz genérica para manejar el resto.

Los zócalos tienen su propio vocabulario:

No Señor.	Plazo y Descripción
1	Dominio La familia de protocolos que se utiliza como mecanismo de transporte. Estos valores son constantes como AF_INET, PF_INET, PF_UNIX, PF_X25, etc.
2	tipo El tipo de comunicaciones entre los dos puntos finales, generalmente SOCK_STREAM para protocolos orientados a conexión y SOCK_DGRAM para protocolos sin conexión.
3	protocolo Típicamente cero, esto puede usarse para identificar una variante de un protocolo dentro de un dominio y tipo.
4 4	nombre de host El identificador de una interfaz de red: <ul style="list-style-type: none">• Una cadena, que puede ser un nombre de host, una dirección de cuatro puntos o una dirección IPV6 en notación de dos puntos (y posiblemente punto)• Una cadena "<castcast>", que especifica una dirección INADDR_BROADCAST.• Una cadena de longitud cero, que especifica INADDR_ANY, o• Un entero, interpretado como una dirección binaria en orden de bytes del host.

5 5

Puerto

Cada servidor escucha a los clientes que llaman a uno o más puertos. Un puerto puede ser un número de puerto Fixnum, una cadena que contiene un número de puerto o el nombre de un servicio.

El zócalo módulo

Para crear un socket, debe usar la función `socket.socket ()` disponible en el módulo de `socket`, que tiene la sintaxis general:

```
s = socket.socket (socket_family, socket_type, protocol=0)
```

Aquí está la descripción de los parámetros:

- **socket_family** : es AF_UNIX o AF_INET, como se explicó anteriormente.
- **socket_type** : es SOCK_STREAM o SOCK_DGRAM.
- **protocolo** : esto generalmente se omite, por defecto a 0.

Una vez que tenga un objeto de `socket`, puede usar las funciones requeridas para crear su programa cliente o servidor. La siguiente es la lista de funciones requeridas:

Métodos de socket del servidor

No Señor.	Método y descripción
1	s.bind () Este método une la dirección (nombre de host, par de número de puerto) al socket.
2	s.listen () Este método configura e inicia el escucha TCP.
3	acepto () Esto acepta pasivamente la conexión del cliente TCP, esperando hasta que llegue la conexión (bloqueo).

Métodos de socket del cliente

No Señor.	Método y descripción
-----------	----------------------

1	<p>s.connect ()</p> <p>Este método inicia activamente la conexión del servidor TCP.</p>
---	--

Métodos generales de socket

No Señor.	Método y descripción
1	<p>s.recv ()</p> <p>Este método recibe un mensaje TCP</p>
2	<p>enviar ()</p> <p>Este método transmite mensajes TCP</p>
3	<p>s.recvfrom ()</p> <p>Este método recibe un mensaje UDP</p>
4 4	<p>s.sendto ()</p> <p>Este método transmite mensajes UDP</p>
5 5	<p>s.close ()</p> <p>Este método cierra el zócalo</p>
6 6	<p>socket.gethostname ()</p> <p>Devuelve el nombre de host.</p>

Un servidor simple

Para escribir servidores de Internet, utilizamos la función de **socket** disponible en el módulo de socket para crear un objeto de socket. Un objeto de socket se usa para llamar a otras funciones para configurar un servidor de socket.

Ahora llame a la función de **enlace (nombre de host, puerto)** para especificar un *puerto* para su servicio en el host dado.

Luego, llame al método `accept` del objeto devuelto. Este método espera hasta que un cliente se conecta al puerto que especificó y luego devuelve un objeto de *conexión* que representa la conexión a ese cliente.

```
#!/usr/bin/python          # This is server.py file

import socket             # Import socket module

s = socket.socket()       # Create a socket object
host = socket.gethostname() # Get local machine name
port = 12345              # Reserve a port for your
service.
s.bind((host, port))     # Bind to the port

s.listen(5)              # Now wait for client connection.
while True:
    c, addr = s.accept()  # Establish connection with
client.
    print 'Got connection from', addr
    c.send('Thank you for connecting')
    c.close()             # Close the connection
```

Un cliente simple

Escribamos un programa cliente muy simple que abra una conexión a un puerto 12345 dado y al host dado. Esto es muy simple para crear un cliente de socket utilizando la función de módulo de `socket` de Python .

El **Socket.connect (hostname, puerto)** abre una conexión TCP con *el nombre de host* en el *puerto* . Una vez que tenga un socket abierto, puede leerlo como cualquier objeto IO. Cuando termine, recuerde cerrarlo, ya que cerraría un archivo.

El siguiente código es un cliente muy simple que se conecta a un host y puerto determinados, lee los datos disponibles del socket y luego sale:

```
#!/usr/bin/python          # This is client.py file

import socket             # Import socket module

s = socket.socket()       # Create a socket object
host = socket.gethostname() # Get local machine name
port = 12345              # Reserve a port for your
service.

s.connect((host, port))
print s.recv(1024)
s.close()                 # Close the socket when done
```

Ahora ejecute este `server.py` en segundo plano y luego ejecute por encima de `client.py` para ver el resultado.

```
# Following would start a server in background.
```

```
$ python server.py &
# Once server is started run client as follows:
$ python client.py
```

Esto produciría el siguiente resultado:

```
Got connection from ('127.0.0.1', 48437)
Thank you for connecting
```

Módulos de internet de Python

Una lista de algunos módulos importantes en la programación de Python Network / Internet.

Protocolo	Función común	Puerto no	Módulo de Python
HTTP	páginas web	80	httplib, urllib, xmlrpclib
NNTP	Noticias Usenet	119	nntplib
FTP	Transferencias de archivos	20	ftplib, urllib
SMTP	Enviando correo electrónico	25	smtplib
POP3	Obteniendo correo electrónico	110	poplib
IMAP4	Obteniendo correo electrónico	143	imaplib
Telnet	Líneas de comando	23	telnetlib
Ardilla de tierra	Transferencias de documentos	70	gopherlib, urllib

Verifique todas las bibliotecas mencionadas anteriormente para trabajar con los protocolos FTP, SMTP, POP e IMAP.

Lecturas adicionales

Este fue un comienzo rápido con la programación de sockets. Es un tema vasto. Se recomienda pasar por el siguiente enlace para encontrar más detalles:

- [Programación de socket Unix](#) .
- [Python Socket Library y Módulos](#) .

Python: envío de correo electrónico mediante SMTP

El Protocolo simple de transferencia de correo (SMTP) es un protocolo que maneja el envío de correo electrónico y el enrutamiento de correo electrónico entre servidores de correo.

Python proporciona el módulo **smtplib** , que define un objeto de sesión de cliente SMTP que se puede usar para enviar correo a cualquier máquina de Internet con un demonio de escucha SMTP o ESMTP.

Aquí hay una sintaxis simple para crear un objeto SMTP, que luego puede usarse para enviar un correo electrónico:

```
import smtplib

smtpObj = smtplib.SMTP( [host [, port [, local_hostname]]] )
```

Aquí está el detalle de los parámetros:

- **host** : este es el host que ejecuta su servidor SMTP. Puede especificar la dirección IP del host o un nombre de dominio como `tutorialspoint.com`. Este es un argumento opcional.
- **puerto** : si proporciona un argumento de *host* , debe especificar un puerto, donde escucha el servidor SMTP. Por lo general, este puerto sería 25.
- **local_hostname** : si su servidor SMTP se está ejecutando en su máquina local, puede especificar solo *localhost* a partir de esta opción.

Un objeto SMTP tiene un método de instancia llamado **sendmail** , que generalmente se usa para hacer el trabajo de enviar un mensaje. Se necesitan tres parámetros:

- El *remitente* : una cadena con la dirección del remitente.
- Los *receptores* : una lista de cadenas, una para cada destinatario.
- El *mensaje* : un mensaje como una cadena formateada como se especifica en los distintos RFC.

Ejemplo

Aquí hay una manera simple de enviar un correo electrónico usando el script Python. Pruébalo una vez

```
#!/usr/bin/python

import smtplib

sender = 'from@fromdomain.com'
receivers = ['to@todomain.com']
```

```
message = """From: From Person <from@fromdomain.com>
To: To Person <to@todomain.com>
Subject: SMTP e-mail test
```

```
This is a test e-mail message.
"""
```

```
try:
    smtpObj = smtplib.SMTP('localhost')
    smtpObj.sendmail(sender, receivers, message)
    print "Successfully sent email"
except SMTPException:
    print "Error: unable to send email"
```

Aquí, ha colocado un correo electrónico básico en el mensaje, utilizando una cita triple, cuidando de formatear los encabezados correctamente. Un correo electrónico requiere un encabezado **De**, **Para** y **Asunto**, separados del cuerpo del correo electrónico con una línea en blanco.

Para enviar el correo, use *smtpObj* para conectarse al servidor SMTP en la máquina local y luego use el método *sendmail* junto con el mensaje, la dirección de origen y la dirección de destino como parámetros (aunque las direcciones de origen y destino estén dentro de -mail en sí, no siempre se usan para enrutar el correo).

Si no está ejecutando un servidor SMTP en su máquina local, puede usar el cliente *smtplib* para comunicarse con un servidor SMTP remoto. A menos que esté utilizando un servicio de correo web (como Hotmail o Yahoo! Mail), su proveedor de correo electrónico debe haberle proporcionado los detalles del servidor de correo saliente que puede proporcionarles, de la siguiente manera:

```
smtplib.SMTP('mail.your-domain.com', 25)
```

Enviar un correo electrónico HTML usando Python

Cuando envía un mensaje de texto usando Python, todo el contenido se trata como texto simple. Incluso si incluye etiquetas HTML en un mensaje de texto, se muestra como texto simple y las etiquetas HTML no se formatearán de acuerdo con la sintaxis HTML. Pero Python ofrece la opción de enviar un mensaje HTML como mensaje HTML real.

Al enviar un mensaje de correo electrónico, puede especificar una versión de Mime, tipo de contenido y juego de caracteres para enviar un correo electrónico HTML.

Ejemplo

El siguiente es el ejemplo para enviar contenido HTML como un correo electrónico. Pruébalo una vez

```
#!/usr/bin/python
```

```

import smtplib

message = """From: From Person <from@fromdomain.com>
To: To Person <to@todomain.com>
MIME-Version: 1.0
Content-type: text/html
Subject: SMTP HTML e-mail test

This is an e-mail message to be sent in HTML format

<b>This is HTML message.</b>
<h1>This is headline.</h1>
"""

try:
    smtpObj = smtplib.SMTP('localhost')
    smtpObj.sendmail(sender, receivers, message)
    print "Successfully sent email"
except SMTPException:
    print "Error: unable to send email"

```

Envío de archivos adjuntos como un correo electrónico

Para enviar un correo electrónico con contenido mixto se requiere establecer el encabezado de **tipo Contenido en multiparte / mixto** . Luego, las secciones de texto y archivo adjunto se pueden especificar dentro de los **límites** .

Se inicia un límite con dos guiones seguidos de un número único, que no puede aparecer en la parte del mensaje del correo electrónico. Un límite final que denota la sección final del correo electrónico también debe terminar con dos guiones.

Los archivos adjuntos deben codificarse con la función de **paquete ("m")** para tener una codificación base64 antes de la transmisión.

Ejemplo

El siguiente es el ejemplo, que envía un archivo **/tmp/test.txt** como archivo adjunto. Pruébalo una vez

```

#!/usr/bin/python

import smtplib
import base64

filename = "/tmp/test.txt"

# Read a file and encode it into base64 format
fo = open(filename, "rb")
filecontent = fo.read()

```

```

encodedcontent = base64.b64encode(filecontent) # base64

sender = 'webmaster@tutorialpoint.com'
reciever = 'amrood.admin@gmail.com'

marker = "AUNIQUEMARKER"

body = """
This is a test email to send an attachement.
"""

# Define the main headers.
part1 = """From: From Person <me@fromdomain.net>
To: To Person <amrood.admin@gmail.com>
Subject: Sending Attachement
MIME-Version: 1.0
Content-Type: multipart/mixed; boundary=%s
--%s
""" % (marker, marker)

# Define the message action
part2 = """Content-Type: text/plain
Content-Transfer-Encoding:8bit

%s
--%s
""" % (body,marker)

# Define the attachment section
part3 = """Content-Type: multipart/mixed; name=\"%s\"
Content-Transfer-Encoding:base64
Content-Disposition: attachment; filename=%s

%s
--%s--
""" % (filename, filename, encodedcontent, marker)
message = part1 + part2 + part3

try:
    smtpObj = smtplib.SMTP('localhost')
    smtpObj.sendmail(sender, reciever, message)
    print "Successfully sent email"
except Exception:
    print "Error: unable to send email"

```

Python - Programación multiproceso

Ejecutar varios subprocesos es similar a ejecutar varios programas diferentes al mismo tiempo, pero con los siguientes beneficios:

- Varios subprocesos dentro de un proceso comparten el mismo espacio de datos con el subproceso principal y, por lo tanto, pueden compartir información o comunicarse entre ellos más fácilmente que si fueran procesos separados.

- Los subprocesos a veces se denominan procesos livianos y no requieren demasiada carga de memoria; Son más baratos que los procesos.

Un hilo tiene un comienzo, una secuencia de ejecución y una conclusión. Tiene un puntero de instrucciones que realiza un seguimiento de dónde se está ejecutando actualmente en su contexto.

- Se puede adelantar (interrumpir)
- Se puede poner temporalmente en espera (también conocido como dormir) mientras se ejecutan otros subprocesos; esto se denomina ceder el paso.

Comenzando un nuevo hilo

Para generar otro hilo, debe llamar al siguiente método disponible en el módulo de *hilo* :

```
thread.start_new_thread ( function, args[, kwargs] )
```

Esta llamada al método permite una forma rápida y eficiente de crear nuevos hilos en Linux y Windows.

La llamada al método regresa inmediatamente y el subproceso hijo comienza y las llamadas funcionan con la lista de *argumentos* aprobada . Cuando la función regresa, el hilo termina.

Aquí, *args* es una tupla de argumentos; use una tupla vacía para llamar a la función sin pasar ningún argumento. *kwargs* es un diccionario opcional de argumentos de palabras clave.

Ejemplo

```
#!/usr/bin/python

import thread
import time

# Define a function for the thread
def print_time( threadName, delay):
    count = 0
    while count < 5:
        time.sleep(delay)
        count += 1
        print "%s: %s" % ( threadName, time.ctime(time.time())
    )

# Create two threads as follows
try:
    thread.start_new_thread( print_time, ("Thread-1", 2, ) )
    thread.start_new_thread( print_time, ("Thread-2", 4, ) )
except:
    print "Error: unable to start thread"

while 1:
    pass
```

Cuando se ejecuta el código anterior, produce el siguiente resultado:

```
Thread-1: Thu Jan 22 15:42:17 2009
Thread-1: Thu Jan 22 15:42:19 2009
Thread-2: Thu Jan 22 15:42:19 2009
Thread-1: Thu Jan 22 15:42:21 2009
Thread-2: Thu Jan 22 15:42:23 2009
Thread-1: Thu Jan 22 15:42:23 2009
Thread-1: Thu Jan 22 15:42:25 2009
Thread-2: Thu Jan 22 15:42:27 2009
Thread-2: Thu Jan 22 15:42:31 2009
Thread-2: Thu Jan 22 15:42:35 2009
```

Aunque es muy efectivo para subprocesos de bajo nivel, el módulo de *subprocesos* es muy limitado en comparación con el módulo de subprocesos más nuevo.

El módulo de *subprocesos*

El módulo de subprocesos más nuevo incluido con Python 2.4 proporciona un soporte de subprocesos mucho más potente que el módulo de subprocesos discutido en la sección anterior.

El módulo de *subprocesos* expone todos los métodos del módulo de *subprocesos* y proporciona algunos métodos adicionales:

- **threading.activeCount ()** : devuelve el número de objetos de subproceso que están activos.
- **threading.currentThread ()** - Devuelve el número de objetos de hilo en el control de hilo del llamador.
- **threading.enumerate ()** - Devuelve una lista de todos los objetos de hilo que están actualmente activos.

Además de los métodos, el módulo de subprocesos tiene la clase *Thread* que implementa subprocesos. Los métodos proporcionados por la clase *Thread* son los siguientes:

- **run ()** : el método run () es el punto de entrada para un hilo.
- **start ()** - El método start () inicia un hilo llamando al método run.
- **join ([time])** : la join () espera a que finalicen los subprocesos.
- **isAlive ()** - El método isAlive () verifica si un hilo todavía se está ejecutando.
- **getName ()** : el método getName () devuelve el nombre de un hilo.
- **setName ()** - El método setName () establece el nombre de un hilo.

Creación de hilos que utilizan *Threading* de módulo

Para implementar un nuevo subproceso utilizando el módulo de subprocesos, debe hacer lo siguiente:

- Defina una nueva subclase de la clase *Thread* .
- Anule el método `__init__ (self [, args])` para agregar argumentos adicionales.

- Luego, anule el método `run (self [, args])` para implementar lo que el hilo debe hacer cuando se inicia.

Una vez que haya creado la nueva subclase de *Thread*, puede crear una instancia de ella y luego iniciar un nuevo thread invocando `start ()`, que a su vez llama al método `run ()`.

Ejemplo

```
#!/usr/bin/python

import threading
import time

exitFlag = 0

class myThread (threading.Thread):
    def __init__(self, threadID, name, counter):
        threading.Thread.__init__(self)
        self.threadID = threadID
        self.name = name
        self.counter = counter
    def run(self):
        print "Starting " + self.name
        print_time(self.name, self.counter, 5)
        print "Exiting " + self.name

def print_time(threadName, counter, delay):
    while counter:
        if exitFlag:
            threadName.exit()
        time.sleep(delay)
        print "%s: %s" % (threadName, time.ctime(time.time()))
        counter -= 1

# Create new threads
thread1 = myThread(1, "Thread-1", 1)
thread2 = myThread(2, "Thread-2", 2)

# Start new Threads
thread1.start()
thread2.start()

print "Exiting Main Thread"
```

Cuando se ejecuta el código anterior, produce el siguiente resultado:

```
Starting Thread-1
Starting Thread-2
Exiting Main Thread
Thread-1: Thu Mar 21 09:10:03 2013
Thread-1: Thu Mar 21 09:10:04 2013
Thread-2: Thu Mar 21 09:10:04 2013
Thread-1: Thu Mar 21 09:10:05 2013
```

```
Thread-1: Thu Mar 21 09:10:06 2013
Thread-2: Thu Mar 21 09:10:06 2013
Thread-1: Thu Mar 21 09:10:07 2013
Exiting Thread-1
Thread-2: Thu Mar 21 09:10:08 2013
Thread-2: Thu Mar 21 09:10:10 2013
Thread-2: Thu Mar 21 09:10:12 2013
Exiting Thread-2
```

Hilos de sincronización

El módulo de subprocesos provisto con Python incluye un mecanismo de bloqueo fácil de implementar que le permite sincronizar subprocesos. Se crea un nuevo bloqueo llamando al método `Lock ()`, que devuelve el nuevo bloqueo.

El método de *adquisición (bloqueo)* del nuevo objeto de bloqueo se utiliza para forzar que los subprocesos se ejecuten sincrónicamente. El parámetro de *bloqueo* opcional le permite controlar si el hilo espera para adquirir el bloqueo.

Si el *bloqueo* se establece en 0, el subproceso vuelve inmediatamente con un valor 0 si no se puede obtener el bloqueo y con un 1 si se adquirió el bloqueo. Si el bloqueo se establece en 1, el hilo se bloquea y espera a que se libere el bloqueo.

El método *release ()* del nuevo objeto de bloqueo se utiliza para liberar el bloqueo cuando ya no es necesario.

Ejemplo

```
#!/usr/bin/python

import threading
import time

class myThread (threading.Thread):
    def __init__(self, threadID, name, counter):
        threading.Thread.__init__(self)
        self.threadID = threadID
        self.name = name
        self.counter = counter
    def run(self):
        print "Starting " + self.name
        # Get lock to synchronize threads
        threadLock.acquire()
        print_time(self.name, self.counter, 3)
        # Free lock to release next thread
        threadLock.release()

def print_time(threadName, delay, counter):
    while counter:
        time.sleep(delay)
```



```

        print "%s: %s" % (threadName, time.ctime(time.time()))
        counter -= 1

threadLock = threading.Lock()
threads = []

# Create new threads
thread1 = myThread(1, "Thread-1", 1)
thread2 = myThread(2, "Thread-2", 2)

# Start new Threads
thread1.start()
thread2.start()

# Add threads to thread list
threads.append(thread1)
threads.append(thread2)

# Wait for all threads to complete
for t in threads:
    t.join()
print "Exiting Main Thread"

```

Cuando se ejecuta el código anterior, produce el siguiente resultado:

```

Starting Thread-1
Starting Thread-2
Thread-1: Thu Mar 21 09:11:28 2013
Thread-1: Thu Mar 21 09:11:29 2013
Thread-1: Thu Mar 21 09:11:30 2013
Thread-2: Thu Mar 21 09:11:32 2013
Thread-2: Thu Mar 21 09:11:34 2013
Thread-2: Thu Mar 21 09:11:36 2013
Exiting Main Thread

```

Cola prioritaria multiproceso

El módulo de *cola* le permite crear un nuevo objeto de cola que puede contener un número específico de elementos. Existen los siguientes métodos para controlar la Cola:

- **get ()** : get () elimina y devuelve un elemento de la cola.
- **put ()** : el put agrega un elemento a una cola.
- **qsize ()** : qsize () devuelve el número de elementos que están actualmente en la cola.
- **empty ()** : el empty () devuelve True si la cola está vacía; de lo contrario, falso.
- **full ()** : full () devuelve True si la cola está llena; de lo contrario, falso.

Ejemplo

```

#!/usr/bin/python

import Queue
import threading
import time

exitFlag = 0

class myThread (threading.Thread):
    def __init__(self, threadID, name, q):
        threading.Thread.__init__(self)
        self.threadID = threadID
        self.name = name
        self.q = q
    def run(self):
        print "Starting " + self.name
        process_data(self.name, self.q)
        print "Exiting " + self.name

def process_data(threadName, q):
    while not exitFlag:
        queueLock.acquire()
        if not workQueue.empty():
            data = q.get()
            queueLock.release()
            print "%s processing %s" % (threadName, data)
        else:
            queueLock.release()
            time.sleep(1)

threadList = ["Thread-1", "Thread-2", "Thread-3"]
nameList = ["One", "Two", "Three", "Four", "Five"]
queueLock = threading.Lock()
workQueue = Queue.Queue(10)
threads = []
threadID = 1

# Create new threads
for tName in threadList:
    thread = myThread(threadID, tName, workQueue)
    thread.start()
    threads.append(thread)
    threadID += 1

# Fill the queue
queueLock.acquire()
for word in nameList:
    workQueue.put(word)
queueLock.release()

# Wait for queue to empty
while not workQueue.empty():
    pass

```

```
# Notify threads it's time to exit
exitFlag = 1

# Wait for all threads to complete
for t in threads:
    t.join()
print "Exiting Main Thread"
```

Cuando se ejecuta el código anterior, produce el siguiente resultado:

```
Starting Thread-1
Starting Thread-2
Starting Thread-3
Thread-1 processing One
Thread-2 processing Two
Thread-3 processing Three
Thread-1 processing Four
Thread-2 processing Five
Exiting Thread-3
Exiting Thread-1
Exiting Thread-2
Exiting Main Thread
```

Python - Procesamiento XML

XML es un lenguaje portátil de código abierto que permite a los programadores desarrollar aplicaciones que pueden ser leídas por otras aplicaciones, independientemente del sistema operativo y / o lenguaje de desarrollo.

¿Qué es el XML?

El Lenguaje de marcado extensible (XML) es un lenguaje de marcado muy parecido a HTML o SGML. Esto lo recomienda el Consorcio World Wide Web y está disponible como un estándar abierto.

XML es extremadamente útil para realizar un seguimiento de pequeñas y medianas cantidades de datos sin requerir una red troncal basada en SQL.

Arquitecturas y API de XML Parser

La biblioteca estándar de Python proporciona un conjunto mínimo pero útil de interfaces para trabajar con XML.

Las dos API para datos XML más básicas y ampliamente utilizadas son las interfaces SAX y DOM.

- **API simple para XML (SAX)**: aquí, registra devoluciones de llamada para eventos de interés y luego deja que el analizador continúe con el documento. Esto es útil cuando sus documentos son grandes o tiene limitaciones de memoria, analiza el archivo a medida que lo lee desde el disco y el archivo completo nunca se almacena en la memoria.
- **API del Modelo de Objetos del Documento (DOM)**: esta es una recomendación del Consorcio de la World Wide Web en la que todo el archivo se lee en la memoria y se almacena en un formulario jerárquico (basado en un árbol) para representar todas las características de un documento XML.

SAX obviamente no puede procesar la información tan rápido como DOM cuando trabaja con archivos grandes. Por otro lado, el uso exclusivo de DOM realmente puede matar sus recursos, especialmente si se usa en muchos archivos pequeños.

SAX es de solo lectura, mientras que DOM permite cambios en el archivo XML. Dado que estas dos API diferentes literalmente se complementan entre sí, no hay ninguna razón por la que no pueda usarlas para proyectos grandes.

Para todos nuestros ejemplos de código XML, usemos un archivo XML simple *movies.xml* como entrada:

```
<collection shelf="New Arrivals">
<movie title="Enemy Behind">
  <type>War, Thriller</type>
  <format>DVD</format>
  <year>2003</year>
  <rating>PG</rating>
  <stars>10</stars>
  <description>Talk about a US-Japan war</description>
</movie>
<movie title="Transformers">
  <type>Anime, Science Fiction</type>
  <format>DVD</format>
  <year>1989</year>
  <rating>R</rating>
  <stars>8</stars>
  <description>A schientific fiction</description>
</movie>
<movie title="Trigun">
  <type>Anime, Action</type>
  <format>DVD</format>
  <episodes>4</episodes>
  <rating>PG</rating>
  <stars>10</stars>
  <description>Vash the Stampede!</description>
```

```
</movie>
<movie title="Ishtar">
  <type>Comedy</type>
  <format>VHS</format>
  <rating>PG</rating>
  <stars>2</stars>
  <description>Viewable boredom</description>
</movie>
</collection>
```

Análisis de XML con API SAX

SAX es una interfaz estándar para el análisis XML controlado por eventos. Analizar XML con SAX generalmente requiere que cree su propio `ContentHandler` subclasificando `xml.sax.ContentHandler`.

Su *ContentHandler* maneja las etiquetas y atributos particulares de su (s) sabor (es) de XML. Un objeto `ContentHandler` proporciona métodos para manejar varios eventos de análisis. Su analizador propietario llama a los métodos `ContentHandler` mientras analiza el archivo XML.

Los métodos *startDocument* y *endDocument* se invocan al inicio y al final del archivo XML. Los *caracteres del método (texto)* pasan datos de caracteres del archivo XML a través del texto del parámetro.

`ContentHandler` se llama al principio y al final de cada elemento. Si el analizador no está en modo de espacio de nombres, se llama a los métodos *startElement* (*etiqueta, atributos*) y *endElement* (*etiqueta*); de lo contrario, se llaman los métodos correspondientes *startElementNS* y *endElementNS*. Aquí, la *etiqueta* es la etiqueta del elemento, y los atributos son un objeto de `Atributos`.

Aquí hay otros métodos importantes para comprender antes de continuar:

El método *make_parser*

El siguiente método crea un nuevo objeto analizador y lo devuelve. El objeto analizador creado será del primer tipo de analizador que encuentre el sistema.

```
xml.sax.make_parser( [parser_list] )
```

Aquí está el detalle de los parámetros:

- **parser_list** : el argumento opcional que consiste en una lista de analizadores para usar que debe implementar el método `make_parser`.

El método de *análisis*

El siguiente método crea un analizador SAX y lo usa para analizar un documento.

```
xml.sax.parse( xmlfile, contenthandler[, errorhandler])
```

Aquí está el detalle de los parámetros:

- **xmlfile** : este es el nombre del archivo XML para leer.
- **contenthandler** : debe ser un objeto ContentHandler.
- **errorhandler** : si se especifica, errorhandler debe ser un objeto SAX ErrorHandler.

El método *parseString*

Hay un método más para crear un analizador SAX y analizar la **cadena XML** especificada .

```
xml.sax.parseString(xmlstring, contenthandler[,  
errorhandler])
```

Aquí está el detalle de los parámetros:

- **xmlstring** : este es el nombre de la cadena XML para leer.
- **contenthandler** : debe ser un objeto ContentHandler.
- **errorhandler** : si se especifica, errorhandler debe ser un objeto SAX ErrorHandler.

Ejemplo

```
#!/usr/bin/python

import xml.sax

class MovieHandler( xml.sax.ContentHandler ):
    def __init__(self):
        self.CurrentData = ""
        self.type = ""
        self.format = ""
        self.year = ""
        self.rating = ""
        self.stars = ""
        self.description = ""

    # Call when an element starts
    def startElement(self, tag, attributes):
        self.CurrentData = tag
        if tag == "movie":
            print "*****Movie*****"
            title = attributes["title"]
            print "Title:", title

    # Call when an elements ends
    def endElement(self, tag):
        if self.CurrentData == "type":
            print "Type:", self.type
        elif self.CurrentData == "format":
            print "Format:", self.format
```

```

elif self.CurrentData == "year":
    print "Year:", self.year
elif self.CurrentData == "rating":
    print "Rating:", self.rating
elif self.CurrentData == "stars":
    print "Stars:", self.stars
elif self.CurrentData == "description":
    print "Description:", self.description
self.CurrentData = ""

# Call when a character is read
def characters(self, content):
    if self.CurrentData == "type":
        self.type = content
    elif self.CurrentData == "format":
        self.format = content
    elif self.CurrentData == "year":
        self.year = content
    elif self.CurrentData == "rating":
        self.rating = content
    elif self.CurrentData == "stars":
        self.stars = content
    elif self.CurrentData == "description":
        self.description = content

if ( __name__ == "__main__" ):

    # create an XMLReader
    parser = xml.sax.make_parser()
    # turn off namespaces
    parser.setFeature(xml.sax.handler.feature_namespaces, 0)

    # override the default ContextHandler
    Handler = MovieHandler()
    parser.setContentHandler( Handler )

    parser.parse("movies.xml")

```

Esto produciría el siguiente resultado:

```

*****Movie*****
Title: Enemy Behind
Type: War, Thriller
Format: DVD
Year: 2003
Rating: PG
Stars: 10
Description: Talk about a US-Japan war
*****Movie*****
Title: Transformers
Type: Anime, Science Fiction
Format: DVD
Year: 1989
Rating: R

```

Stars: 8
Description: A schientific fiction
*****Movie*****
Title: Trigun
Type: Anime, Action
Format: DVD
Rating: PG
Stars: 10
Description: Vash the Stampede!
*****Movie*****
Title: Ishtar
Type: Comedy
Format: VHS
Rating: PG
Stars: 2
Description: Viewable boredom

Para obtener detalles completos sobre la documentación de la API SAX, consulte las [API SAX Python](#) estándar .

Análisis de XML con API de DOM

El Document Object Model ("DOM") es una API en varios idiomas del World Wide Web Consortium (W3C) para acceder y modificar documentos XML.

El DOM es extremadamente útil para aplicaciones de acceso aleatorio. SAX solo le permite ver un bit del documento a la vez. Si está mirando un elemento SAX, no tiene acceso a otro.

Aquí está la forma más fácil de cargar rápidamente un documento XML y crear un objeto minidom usando el módulo xml.dom. El objeto minidom proporciona un método de analizador simple que crea rápidamente un árbol DOM a partir del archivo XML.

La frase de ejemplo llama a la función de análisis (archivo [, analizador]) del objeto minidom para analizar el archivo XML designado por archivo en un objeto de árbol DOM.

```
#!/usr/bin/python

from xml.dom.minidom import parse
import xml.dom.minidom

# Open XML document using minidom parser
DOMTree = xml.dom.minidom.parse("movies.xml")
collection = DOMTree.documentElement
if collection.hasAttribute("shelf"):
    print "Root element : %s" %
collection.getAttribute("shelf")

# Get all the movies in the collection
movies = collection.getElementsByTagName("movie")

# Print detail of each movie.
```



```

for movie in movies:
    print "*****Movie*****"
    if movie.hasAttribute("title"):
        print "Title: %s" % movie.getAttribute("title")

    type = movie.getElementsByTagName('type')[0]
    print "Type: %s" % type.childNodes[0].data
    format = movie.getElementsByTagName('format')[0]
    print "Format: %s" % format.childNodes[0].data
    rating = movie.getElementsByTagName('rating')[0]
    print "Rating: %s" % rating.childNodes[0].data
    description = movie.getElementsByTagName('description')[0]
    print "Description: %s" % description.childNodes[0].data

```

Esto produciría el siguiente resultado:

```

Root element : New Arrivals
*****Movie*****
Title: Enemy Behind
Type: War, Thriller
Format: DVD
Rating: PG
Description: Talk about a US-Japan war
*****Movie*****
Title: Transformers
Type: Anime, Science Fiction
Format: DVD
Rating: R
Description: A schientific fiction
*****Movie*****
Title: Trigun
Type: Anime, Action
Format: DVD
Rating: PG
Description: Vash the Stampede!
*****Movie*****
Title: Ishtar
Type: Comedy
Format: VHS
Rating: PG
Description: Viewable boredom

```

Para obtener detalles completos sobre la documentación de DOM API, consulte las [API estándar de Python DOM](#).

Python - Programación GUI (Tkinter)

Python proporciona varias opciones para desarrollar interfaces gráficas de usuario (GUI). Los más importantes se enumeran a continuación.

- **Tkinter** : Tkinter es la interfaz de Python para el kit de herramientas Tk GUI que se incluye con Python. Buscaríamos esta opción en este capítulo.
- **wxPython** : esta es una interfaz de Python de código abierto para wxWindows <http://wxpython.org>.

- **JPython** : JPython es un puerto de Python para Java que proporciona a los scripts de Python un acceso perfecto a las bibliotecas de clases de Java en la máquina local <http://www.jython.org> .

Hay muchas otras interfaces disponibles, que puede encontrar en la red.

Programación Tkinter

Tkinter es la biblioteca GUI estándar para Python. Python cuando se combina con Tkinter proporciona una manera rápida y fácil de crear aplicaciones GUI. Tkinter proporciona una potente interfaz orientada a objetos para el kit de herramientas Tk GUI.

Crear una aplicación GUI usando Tkinter es una tarea fácil. Todo lo que necesita hacer es realizar los siguientes pasos:

- Importe el módulo *Tkinter* .
- Cree la ventana principal de la aplicación GUI.
- Agregue uno o más de los widgets mencionados anteriormente a la aplicación GUI.
- Ingrese el bucle principal del evento para tomar medidas contra cada evento activado por el usuario.

Ejemplo

```
#!/usr/bin/python

import Tkinter
top = Tkinter.Tk()
# Code to add widgets will go here...
top.mainloop()
```

Esto crearía una siguiente ventana:



Tkinter Widgets

Tkinter proporciona varios controles, como botones, etiquetas y cuadros de texto utilizados en una aplicación GUI. Estos controles se denominan comúnmente widgets.

Actualmente hay 15 tipos de widgets en Tkinter. Presentamos estos widgets, así como una breve descripción en la siguiente tabla:

No Señor.	Operador y Descripción
1	<u>Botón</u> El widget Botón se usa para mostrar botones en su aplicación.
2	<u>Lona</u> El widget Canvas se usa para dibujar formas, como líneas, óvalos, polígonos y rectángulos, en su aplicación.
3	<u>Botón de control</u> El widget Checkbutton se usa para mostrar una serie de opciones como casillas de verificación. El usuario puede seleccionar múltiples opciones a la vez.
4 4	<u>Entrada</u> El widget Entrada se usa para mostrar un campo de texto de una sola línea para aceptar valores de un usuario.
5 5	<u>Marco</u> El widget Frame se usa como widget contenedor para organizar otros widgets.
6 6	<u>Etiqueta</u> El widget de etiqueta se utiliza para proporcionar un título de línea única para otros widgets. También puede contener imágenes.
7 7	<u>Cuadro de lista</u> El widget Listbox se usa para proporcionar una lista de opciones a un usuario.
8	<u>Botón de menú</u> El widget Menubutton se usa para mostrar menús en su aplicación.
9 9	<u>Menú</u> El widget Menú se usa para proporcionar varios comandos a un usuario. Estos comandos están contenidos dentro de Menubutton.
10	<u>Mensaje</u> El widget Mensaje se usa para mostrar campos de texto de varias líneas para

	aceptar valores de un usuario.
11	<p><u>Boton de radio</u></p> <p>El widget Radiobutton se usa para mostrar una serie de opciones como botones de radio. El usuario puede seleccionar solo una opción a la vez.</p>
12	<p><u>Escala</u></p> <p>El widget Escala se usa para proporcionar un widget deslizante.</p>
13	<p><u>Barra de desplazamiento</u></p> <p>El widget Barra de desplazamiento se usa para agregar capacidad de desplazamiento a varios widgets, como cuadros de lista.</p>
14	<p><u>Texto</u></p> <p>El widget de texto se usa para mostrar texto en varias líneas.</p>
15	<p><u>Nivel superior</u></p> <p>El widget Toplevel se usa para proporcionar un contenedor de ventana separado.</p>
dieciséis	<p><u>Spinbox</u></p> <p>El widget Spinbox es una variante del widget Tkinter Entry estándar, que se puede usar para seleccionar entre un número fijo de valores.</p>
17	<p><u>PanedWindow</u></p> <p>Un PanedWindow es un widget contenedor que puede contener cualquier número de paneles, dispuestos horizontal o verticalmente.</p>
18 años	<p><u>LabelFrame</u></p> <p>Un labelframe es un widget contenedor simple. Su propósito principal es actuar como un espaciador o contenedor para diseños complejos de ventanas.</p>
19	<p><u>tkMessageBox</u></p> <p>Este módulo se utiliza para mostrar cuadros de mensaje en sus aplicaciones.</p>

Estudiamos estos widgets en detalle:

Atributos estándar

Echemos un vistazo a cómo se especifican algunos de sus atributos comunes, como los tamaños, los colores y las fuentes.

- Dimensiones
- Colores
- Fuentes

- [Anclas](#)
- [Estilos de relieve](#)
- [Mapas de bits](#)
- [Cursores](#)

Vamos a estudiarlos brevemente.

Geometry Management

Todos los widgets de Tkinter tienen acceso a métodos específicos de administración de geometría, que tienen el propósito de organizar los widgets en toda el área de widgets principal. Tkinter expone las siguientes clases de administrador de geometría: paquete, cuadrícula y lugar.

- [El `pack\(\)` Método](#) - Este gestor de la geometría organiza widgets en bloques antes de colocarlos en el widget padre.
- [El método `grid\(\)`](#): este administrador de geometría organiza widgets en una estructura similar a una tabla en el widget principal.
- [El método `place\(\)`](#): este administrador de geometría organiza los widgets colocándolos en una posición específica en el widget principal.

Estudiamos brevemente los métodos de gestión de la geometría.

Python - Programación de extensiones con C

Cualquier código que escriba utilizando cualquier lenguaje compilado como C, C ++ o Java puede integrarse o importarse a otro script de Python. Este código se considera como una "extensión".

Un módulo de extensión de Python no es más que una biblioteca C normal. En máquinas Unix, estas bibliotecas generalmente terminan en **.so** (para objetos compartidos). En las máquinas con Windows, normalmente ve **.dll** (para la biblioteca vinculada dinámicamente).

Requisitos previos para escribir extensiones

Para comenzar a escribir su extensión, necesitará los archivos de encabezado de Python.

- En las máquinas Unix, esto generalmente requiere la instalación de un paquete específico para el desarrollador, como [python2.5-dev](#) .

- Los usuarios de Windows obtienen estos encabezados como parte del paquete cuando usan el instalador binario de Python.

Además, se supone que tiene un buen conocimiento de C o C ++ para escribir cualquier extensión de Python utilizando la programación en C.

Primer vistazo a una extensión de Python

Para su primer vistazo a un módulo de extensión de Python, debe agrupar su código en cuatro partes:

- El archivo de encabezado *Python.h* .
- Las funciones C que desea exponer como la interfaz de su módulo.
- Una tabla que asigna los nombres de sus funciones a medida que los desarrolladores de Python las ven a las funciones C dentro del módulo de extensión.
- Una función de inicialización.

El archivo de encabezado *Python.h*

Debe incluir el archivo de encabezado *Python.h* en su archivo fuente C, que le da acceso a la API interna de Python utilizada para conectar su módulo al intérprete.

Asegúrese de incluir *Python.h* antes que cualquier otro encabezado que pueda necesitar. Debe seguir las inclusiones con las funciones a las que desea llamar desde Python.

Las funciones C

Las firmas de la implementación en C de sus funciones siempre toman una de las siguientes tres formas:

```
static PyObject *MyFunction( PyObject *self, PyObject *args
);

static PyObject *MyFunctionWithKeywords(PyObject *self,
                                       PyObject *args,
                                       PyObject *kw);

static PyObject *MyFunctionWithNoArgs( PyObject *self );
```

Cada una de las declaraciones anteriores devuelve un objeto Python. No existe una función *nula* en Python como la hay en C. Si no desea que sus funciones devuelvan un valor, devuelva el equivalente en C del valor **None** de Python. Los encabezados de Python definen una macro, `Py_RETURN_NONE`, que hace esto por nosotros.

Los nombres de sus funciones de C pueden ser lo que quiera, ya que nunca se ven fuera del módulo de extensión. Se definen como función *estática* .

Sus funciones C generalmente se nombran combinando el módulo Python y los nombres de las funciones, como se muestra aquí:

```
static PyObject *module_func(PyObject *self, PyObject *args)
{
    /* Do your stuff here. */
    Py_RETURN_NONE;
}
```

Esta es una función de Python llamada *func* dentro del módulo del *módulo*. Pondrá punteros a sus funciones C en la tabla de métodos para el módulo que generalmente viene después en su código fuente.

La tabla de mapeo de métodos

Esta tabla de métodos es una matriz simple de estructuras PyMethodDef. Esa estructura se parece a esto:

```
struct PyMethodDef {
    char *ml_name;
    PyCFunction ml_meth;
    int ml_flags;
    char *ml_doc;
};
```

Aquí está la descripción de los miembros de esta estructura:

- **ml_name** : este es el nombre de la función que el intérprete de Python presenta cuando se usa en programas de Python.
- **ml_meth** : debe ser la dirección de una función que tenga cualquiera de las firmas descritas en la sección anterior.
- **ml_flags** : le dice al intérprete cuál de las tres firmas está usando ml_meth.
 - Esta bandera generalmente tiene un valor de METH_VARARGS.
 - Este indicador se puede OR a bit con METH_KEYWORDS si desea permitir argumentos de palabras clave en su función.
 - Esto también puede tener un valor de METH_NOARGS que indica que no desea aceptar ningún argumento.
- **ml_doc** : esta es la cadena de documentación de la función, que podría ser NULL si no tiene ganas de escribir una.

Esta tabla debe terminarse con un centinela que consta de valores NULL y 0 para los miembros apropiados.

Ejemplo

Para la función definida anteriormente, tenemos la siguiente tabla de mapeo de métodos:

```
static PyMethodDef module_methods[] = {
    { "func", (PyCFunction)module_func, METH_NOARGS, NULL },
    { NULL, NULL, 0, NULL }
```

```
};
```

La función de inicialización

La última parte de su módulo de extensión es la función de inicialización. El intérprete de Python llama a esta función cuando se carga el módulo. Se requiere que la función se llame **init *Module***, donde *Module* es el nombre del módulo.

La función de inicialización debe exportarse desde la biblioteca que construirá. Los encabezados de Python definen `PyMODINIT_FUNC` para incluir los encantamientos apropiados para que eso suceda para el entorno particular en el que estamos compilando. Todo lo que tiene que hacer es usarlo al definir la función.

Su función de inicialización de C generalmente tiene la siguiente estructura general:

```
PyMODINIT_FUNC initModule() {  
    Py_InitModule3(func, module_methods, "docstring...");  
}
```

Aquí está la descripción de la función `Py_InitModule3`:

- **func**: esta es la función que se exportará.
- **module_methods**: este es el nombre de la tabla de asignación definida anteriormente.
- **docstring**: este es el comentario que desea dar en su extensión.

Poniendo todo esto junto se ve así:

```
#include <Python.h>  
  
static PyObject *module_func(PyObject *self, PyObject *args)  
{  
    /* Do your stuff here. */  
    Py_RETURN_NONE;  
}  
  
static PyMethodDef module_methods[] = {  
    { "func", (PyCFunction)module_func, METH_NOARGS, NULL },  
    { NULL, NULL, 0, NULL }  
};  
  
PyMODINIT_FUNC initModule() {  
    Py_InitModule3(func, module_methods, "docstring...");  
}
```

Ejemplo

Un ejemplo simple que hace uso de todos los conceptos anteriores:

```
#include <Python.h>
```



```

static PyObject* helloworld(PyObject* self) {
    return Py_BuildValue("s", "Hello, Python extensions!!");
}

static char helloworld_docs[] =
    "helloworld( ): Any message you want to put here!!\n";

static PyMethodDef helloworld_funcs[] = {
    {"helloworld", (PyCFunction)helloworld,
     METH_NOARGS, helloworld_docs},
    {NULL}
};

void inithelloworld(void) {
    Py_InitModule3("helloworld", helloworld_funcs,
                  "Extension module example!");
}

```

Aquí la función *Py_BuildValue* se usa para construir un valor de Python. Guarde el código anterior en el archivo `hello.c`. Veríamos cómo compilar e instalar este módulo para que se llame desde el script Python.

Construyendo e Instalando Extensiones

El paquete *distutils* hace que sea muy fácil distribuir módulos Python, tanto Python puro como módulos de extensión, de una manera estándar. Los módulos se distribuyen en forma de origen y se *compilan* e instalan a través de un script de configuración que generalmente se denomina *setup.py* de la siguiente manera.

Para el módulo anterior, debe preparar el siguiente script `setup.py`:

```

from distutils.core import setup, Extension
setup(name='helloworld', version='1.0', \
      ext_modules=[Extension('helloworld', ['hello.c'])])

```

Ahora, use el siguiente comando, que realizaría todos los pasos necesarios de compilación y vinculación, con los comandos e indicadores del compilador y vinculador correctos, y copia la biblioteca dinámica resultante en un directorio apropiado:

```
$ python setup.py install
```

En los sistemas basados en Unix, lo más probable es que necesite ejecutar este comando como root para tener permisos para escribir en el directorio de paquetes del sitio. Esto generalmente no es un problema en Windows.

Importando Extensiones

Una vez que haya instalado su extensión, podrá importar y llamar a esa extensión en su script de Python de la siguiente manera:

```
#!/usr/bin/python
import helloworld

print helloworld.helloworld()
```

Esto produciría el siguiente resultado:

```
Hello, Python extensions!!
```

Parámetros de funciones de paso

Como lo más probable es que desee definir funciones que acepten argumentos, puede usar una de las otras firmas para sus funciones C. Por ejemplo, la siguiente función, que acepta cierto número de parámetros, se definiría así:

```
static PyObject *module_func(PyObject *self, PyObject *args)
{
    /* Parse args and do something interesting here. */
    Py_RETURN_NONE;
}
```

La tabla de métodos que contiene una entrada para la nueva función se vería así:

```
static PyMethodDef module_methods[] = {
    { "func", (PyCFunction)module_func, METH_NOARGS, NULL },
    { "func", module_func, METH_VARARGS, NULL },
    { NULL, NULL, 0, NULL }
};
```

Puede usar la función API `PyArg_ParseTuple` para extraer los argumentos del puntero `PyObject` pasado a su función C.

El primer argumento para `PyArg_ParseTuple` es el argumento `args`. Este es el objeto que *analizará*. El segundo argumento es una cadena de formato que describe los argumentos tal como espera que aparezcan. Cada argumento está representado por uno o más caracteres en la cadena de formato de la siguiente manera.

```
static PyObject *module_func(PyObject *self, PyObject *args)
{
    int i;
    double d;
    char *s;

    if (!PyArg_ParseTuple(args, "ids", &i, &d, &s)) {
        return NULL;
    }

    /* Do something interesting here. */
    Py_RETURN_NONE;
}
```

Compilar la nueva versión de su módulo e importarlo le permite invocar la nueva función con cualquier número de argumentos de cualquier tipo:

```
module.func(1, s="three", d=2.0)
module.func(i=1, d=2.0, s="three")
module.func(s="three", d=2.0, i=1)
```

Probablemente puedas encontrar aún más variaciones.

La función *PyArg_ParseTuple*

Aquí está la firma estándar para la función **PyArg_ParseTuple** :

```
int PyArg_ParseTuple(PyObject* tuple, char* format, ...)
```

Esta función devuelve 0 para errores y un valor no igual a 0 para el éxito. *tupla* es el `PyObject *` que fue el segundo argumento de la función C. Aquí el *formato* es una cadena C que describe argumentos obligatorios y opcionales.

Aquí hay una lista de códigos de formato para la función **PyArg_ParseTuple** :

Código	Tipo C	Sentido
C	carbonizarse	Una cadena de Python de longitud 1 se convierte en C char.
re	doble	Un flotador de Python se convierte en un doble C.
F	flotador	Un flotador Python se convierte en un flotador C.
yo	En t	Un Python int se convierte en un C int.
l	largo	Un Python int se convierte en una C larga.
L	largo largo	Un Python int se convierte en un C largo largo
O	PyObject *	Obtiene una referencia prestada no NULL al argumento de Python.
s	carbonizarse*	Cadena de Python sin nullos incrustados en C char *.
s #	char * + int	Cualquier cadena de Python a dirección C y longitud.

t #	char * + int	Búfer de solo segmento de solo lectura para dirección C y longitud.
tu	Py_UNICODE *	Python Unicode sin nulos incrustados en C.
u #	Py_UNICODE * + int	Cualquier dirección y longitud de Python Unicode C.
w #	char * + int	Lectura / escritura de búfer de segmento único en dirección C y longitud.
z	carbonizarse*	Al igual que s, también acepta None (establece C char * en NULL).
z #	char * + int	Al igual que s #, también acepta None (establece C char * en NULL).
(...)	según ...	Una secuencia de Python se trata como un argumento por elemento.
El		Los siguientes argumentos son opcionales.
:		Fin del formato, seguido del nombre de la función para los mensajes de error.
;		Fin del formato, seguido de todo el texto del mensaje de error.

Valores devueltos

Py_BuildValue toma una cadena de formato muy similar a *PyArg_ParseTuple*. En lugar de pasar las direcciones de los valores que está creando, pasa los valores reales. Aquí hay un ejemplo que muestra cómo implementar una función de agregar:

```
static PyObject *foo_add(PyObject *self, PyObject *args) {
    int a;
    int b;

    if (!PyArg_ParseTuple(args, "ii", &a, &b)) {
        return NULL;
    }
}
```

```
return Py_BuildValue("i", a + b);
}
```

Así es como se vería si se implementara en Python:

```
def add(a, b):
    return (a + b)
```

Puede devolver dos valores de su función de la siguiente manera, esto se capturaría utilizando una lista en Python.

```
static PyObject *foo_add_subtract(PyObject *self, PyObject
*args) {
    int a;
    int b;

    if (!PyArg_ParseTuple(args, "ii", &a, &b)) {
        return NULL;
    }
    return Py_BuildValue("ii", a + b, a - b);
}
```

Así es como se vería si se implementara en Python:

```
def add_subtract(a, b):
    return (a + b, a - b)
```

La función *Py_BuildValue*

Aquí está la firma estándar para la función **Py_BuildValue** :

```
PyObject* Py_BuildValue(char* format, ...)
```

Aquí el *formato* es una cadena C que describe el objeto Python a construir. Los siguientes argumentos de *Py_BuildValue* son valores C a partir de los cuales se construye el resultado. El resultado de *PyObject ** es una nueva referencia.

La siguiente tabla enumera las cadenas de código de uso común, de las cuales cero o más se unen en formato de cadena.

Código	Tipo C	Sentido
C	carbonizarse	AC char se convierte en una cadena Python de longitud 1.
re	doble	AC doble se convierte en un flotador de Python.
F	flotador	El flotador AC se convierte en un flotador Python.
yo	En t	AC int se convierte en Python int.

I	largo	AC se convierte en Python int.
norte	PyObject *	Pasa un objeto Python y roba una referencia.
O	PyObject *	Pasa un objeto de Python y lo INCREMENTA de forma normal.
O y	convertir + anular *	Conversión arbitraria
s	carbonizarse*	Char * terminado en C 0 a una cadena de Python, o NULL a None.
s #	char * + int	C char * y longitud a Python string, o NULL a None.
tu	Py_UNICODE *	C-wide, cadena terminada en nulo en Python Unicode, o NULL en None.
u #	Py_UNICODE * + int	Cadena de ancho C y longitud a Python Unicode, o NULL a None.
w #	char * + int	Lectura / escritura de búfer de segmento único en dirección C y longitud.
z	carbonizarse*	Al igual que s, también acepta None (establece C char * en NULL).
z #	char * + int	Al igual que s #, también acepta None (establece C char * en NULL).
(...)	según ...	Construye la tupla de Python a partir de los valores de C.
[...]	según ...	Crea una lista de Python a partir de los valores de C.
{...}	según ...	Construye el diccionario Python a partir de valores C, alternando claves y valores.

El código {...} crea diccionarios a partir de un número par de valores C, alternativamente claves y valores. Por ejemplo, Py_BuildValue ("{issi}", 23, "zig", "zag", 42) devuelve un diccionario como {23: 'zig', 'zag': 42} de Python.

Descarga más libros de programación GRATIS [click aquí](#)



Síguenos en Instagram para que estés al tanto de los nuevos libros de programación. [Click aquí](#)