Python Básico Para Hackers y Pentester

Aprende a desarrollar tus propias herramientas con Python desde cero.

Autor: Sebastián Veliz Donoso

“Este libro es libre, no se compra ni se vende”

Edición Número 1.0

Contenido

Palabras del Autor

Este libro está enfocado principalmente a hacker que necesiten aprender Python desde lo más básico, con el fin de tener los fundamentos necesarios para escribir sus propias herramientas o modificar las ya escritas. Todos los ejemplos vistos en este libro, son evaluados desde el punto de vista hacker para hacer más fácil su lectura e interpretación.

Finalmente, espero que este humilde libro sea de su gusto y les ayude a crecer como profesionales de CiberSeguridad.

Este libro está permitido piratearlo y compartirlo cuantas veces quieran, lo único que pido es que si utilizas mis ejemplos en sus clases, charlas o artículos, dejes una cita desde donde sacaron la información.

Happy Hacking!

Preparando el entorno para comenzar a programar

01

Lenguaje de Programación Python

Phyton es un lenguaje de programación fácil de aprender, potente, sencillo, que incluso algunas veces puede ser confundido con pseudo código. Dentro de sus características principales podemos destacar que; es un lenguaje interpretado, tiene múltiples librerías instaladas por defecto, no es necesario declarar el tipo de dato que va a contener una determinada variable, el lenguaje entrega la posibilidad de escribir código orientada a objetos o estructurado según el gusto del programador, su sintaxis es elegante y sencilla, el código es portable y soportado por múltiples plataformas. Todas estas características convierten a Python en un lenguaje ideal para hackers, que tengan la necesidad de crear herramientas especializadas.

Guido van Rossum es un científico de la computación, conocido por ser el autor del lenguaje de programación Python.

Iniciándome en Python

Para comenzar, debemos saber que Python viene instalado por defecto en la mayoría de los sistemas operativos (Linux, Mac Os, Solaris y AIX). El único sistema operativo que no cuenta Python instalado por defecto es Windows, pero si gustas de usar este sistema operativo, debes descargar el instalador Windows, hacer doble clic y hacer siguiente, siguiente a todo.

Todos los ejemplos que vamos a ver en este libro, van a ser realizados en una maquina con sistema Operativo Linux, por lo que recomiendo que si no tienes instalado Linux, te instales una maquina virtual con este sistema operativo.

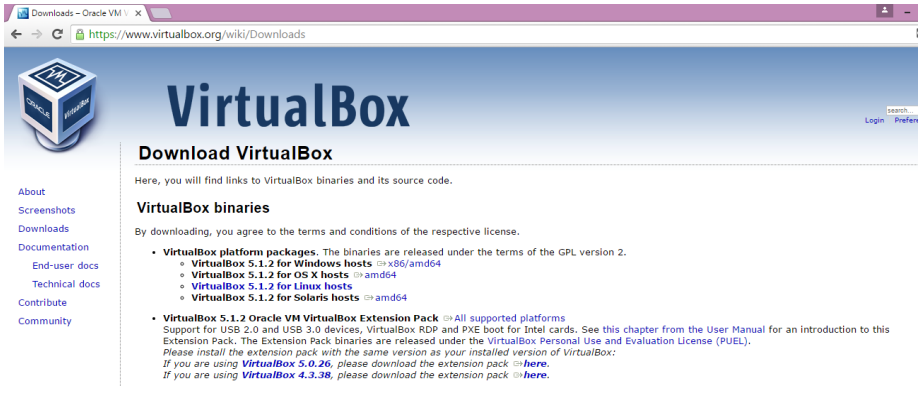
Preparando el ambiente para comenzar a programar

En el caso de que no tengamos instalado Linux por defecto en nuestro computador, necesitaremos tener instalado:

1- Virtual box
2- Ubuntu 14.04.2 LTS

Para la instalación de Virtual box vamos a ingresar al siguiente link en nuestro Browser:

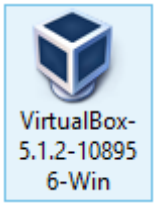
https://www.virtualbox.org/wiki/Downloads



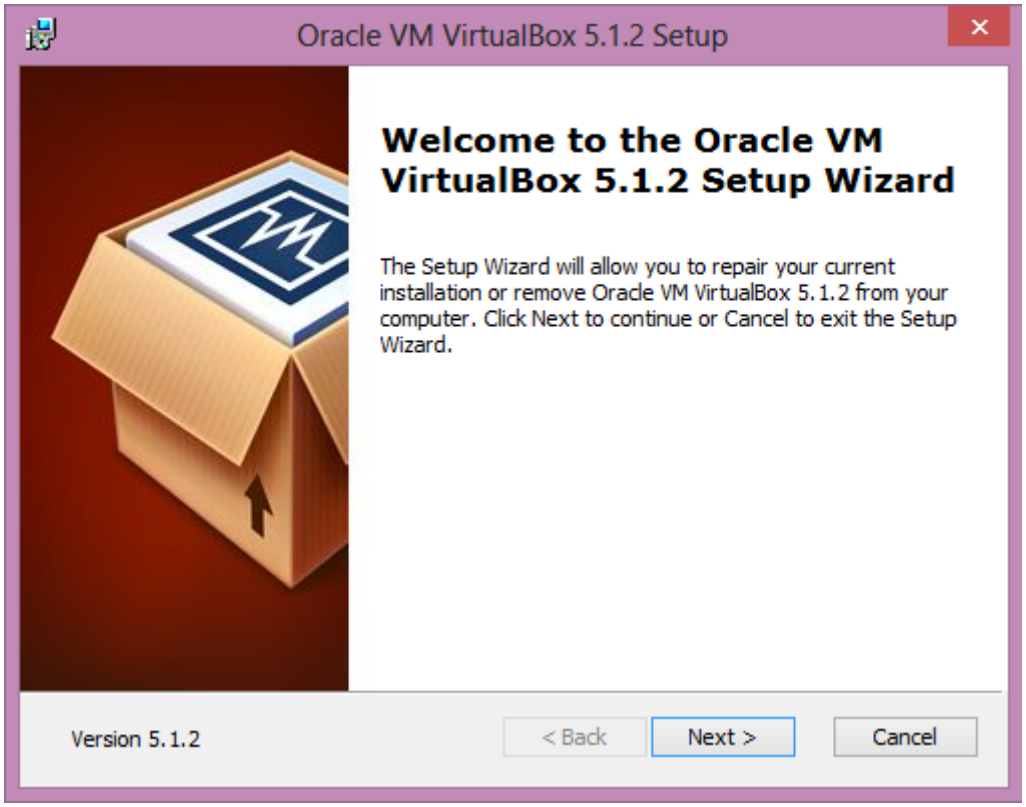
En mi caso, voy a descargar la versión x86/amd64

- **VirtualBox platform packages**. The binaries are released under the terms of the GPL version 2.
  - **VirtualBox 5.1.2 for Windows hosts** ⇨x86/amd64
  - **VirtualBox 5.1.2 for OS X hosts** ⇨amd64
  - **VirtualBox 5.1.2 for Linux hosts**
  - **VirtualBox 5.1.2 for Solaris hosts** ⇨amd64

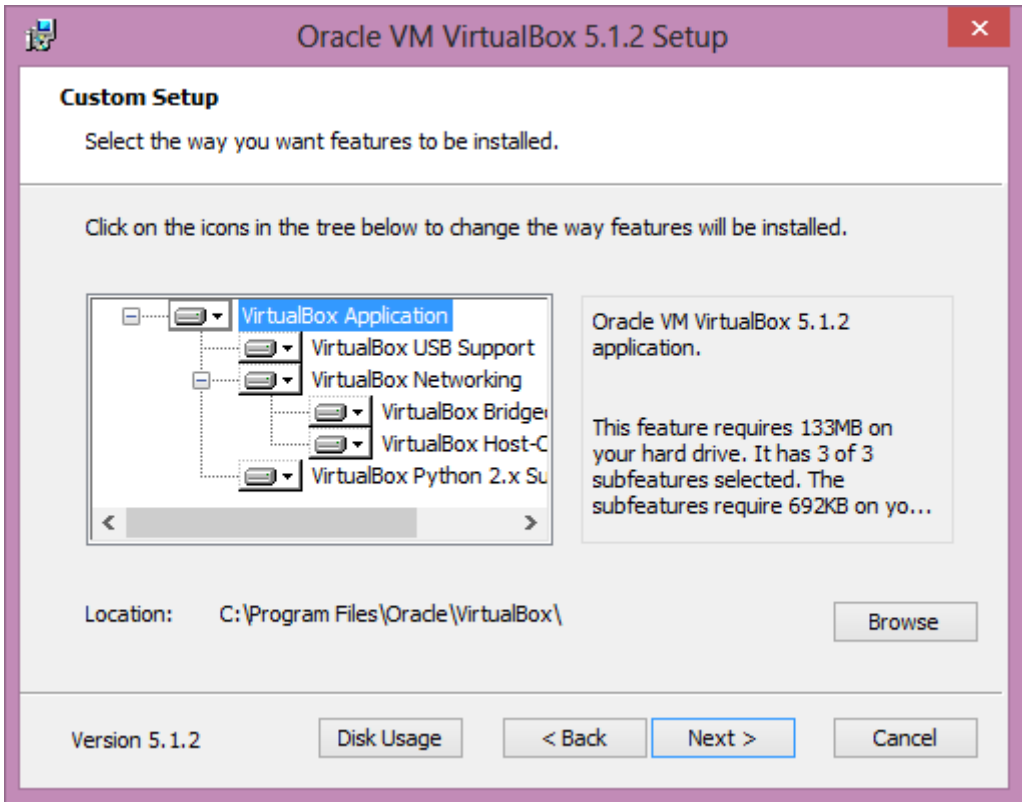
Una vez descargado el Instalador, procedemos a dar doble clic sobre el icono.



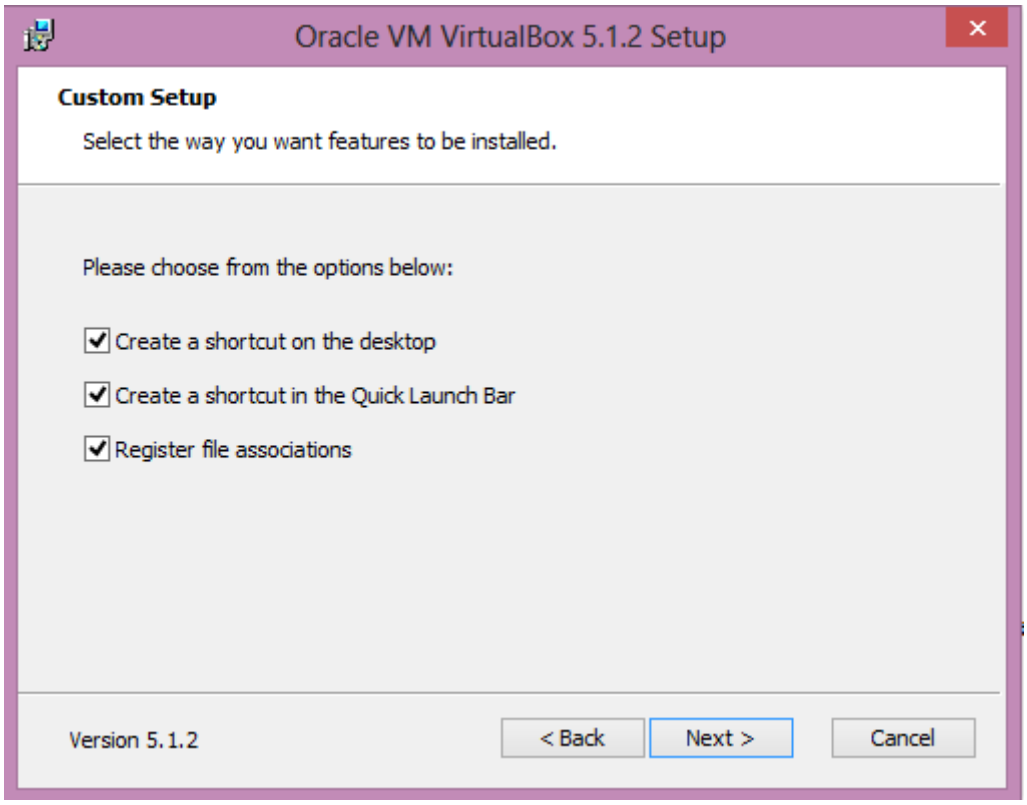
Y se nos abrirá en pantalla el asistente de instalación, donde procedemos a dar clic en el botón Next



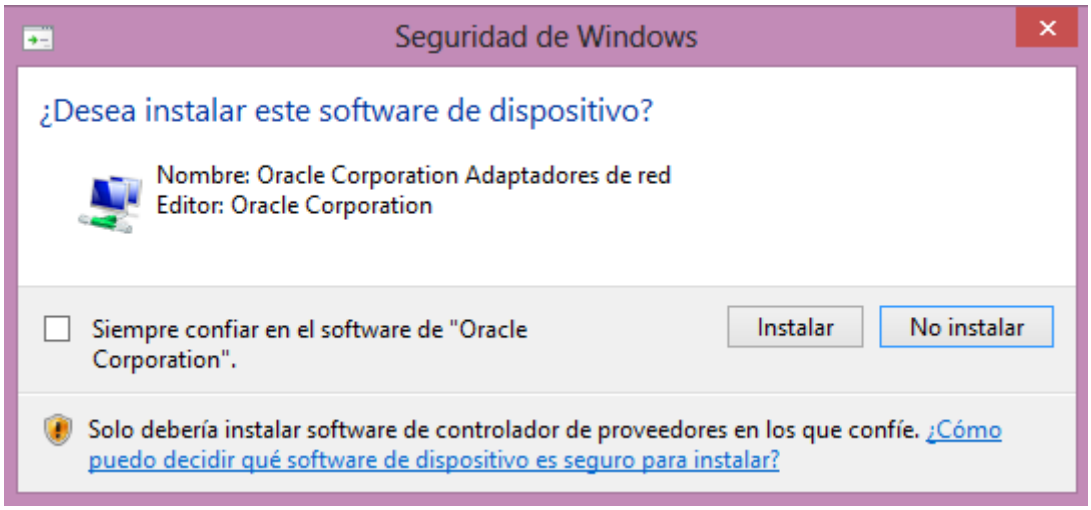
Se nos pedirá en pantalla si deseamos personalizar nuestra instalación, para este ejemplo voy a dejar todo por defecto y doy click al botón Next.



Nuevamente damos click al botón Next



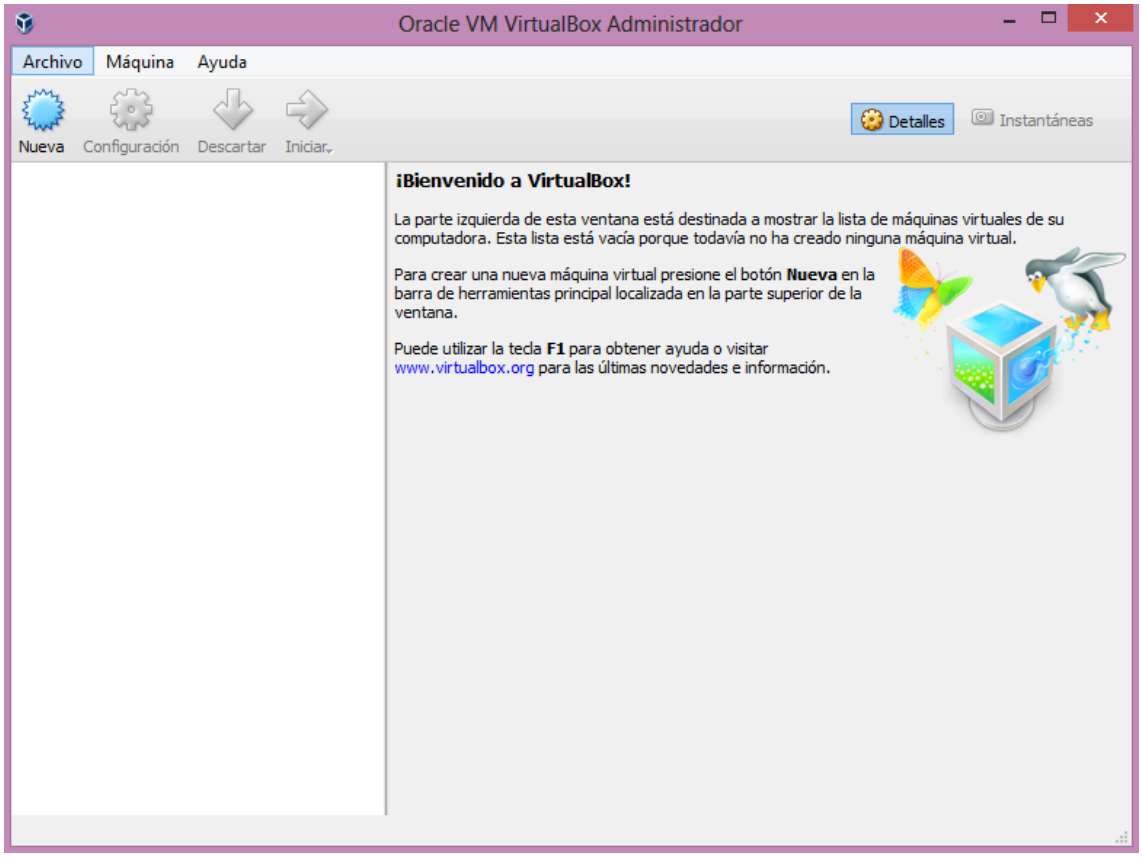
El asistente de instalación pregunta si queremos instalar algunos controladores. Para que nuestra maquina virtual funcione correctamente le damos click a Instalar.



Estos mensajes podrían aparecer varias veces a la instalación por lo que se recomienda instalar todo lo que se nos solicite.

Finalmente nos aparecerá en pantalla que el programa se instalo correctamente, damos click a Finish.

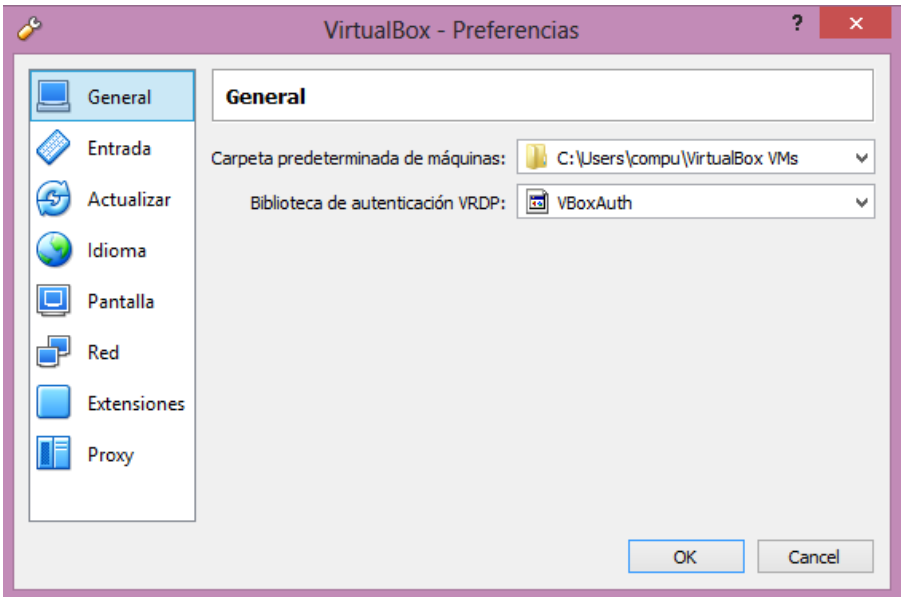
Si nuestra instalación sale bien, deberíamos tener la siguiente pantalla.



Como complemento extra a la instalación de Virtual box, recomiendo instalar también las extensiones que se encuentran en el sitio de descargas. Para esto damos click a el link All supported platforms y descargamos.

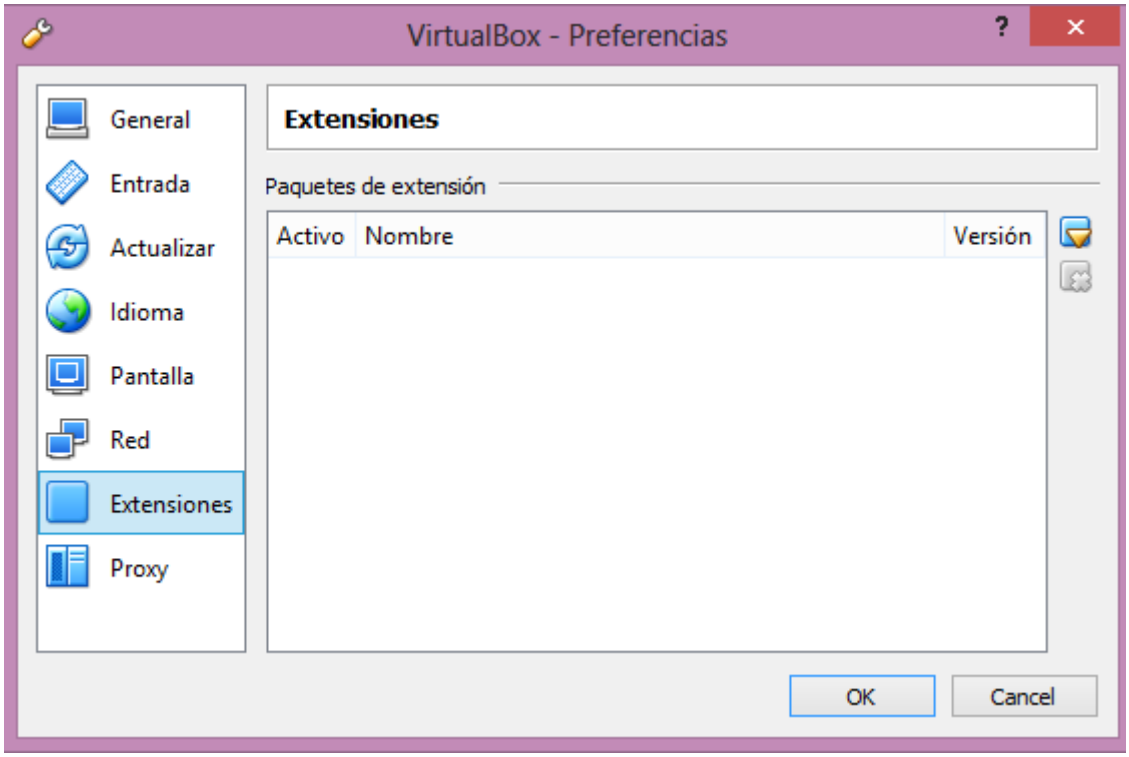
- **VirtualBox 5.1.2 Oracle VM VirtualBox Extension Pack** All supported platforms

Una vez lista la descarga, vamos a nuestra maquina virtual y damos click en el Menú: Archivos > Preferencias…



Vamos al menú donde dice Extensiones

.

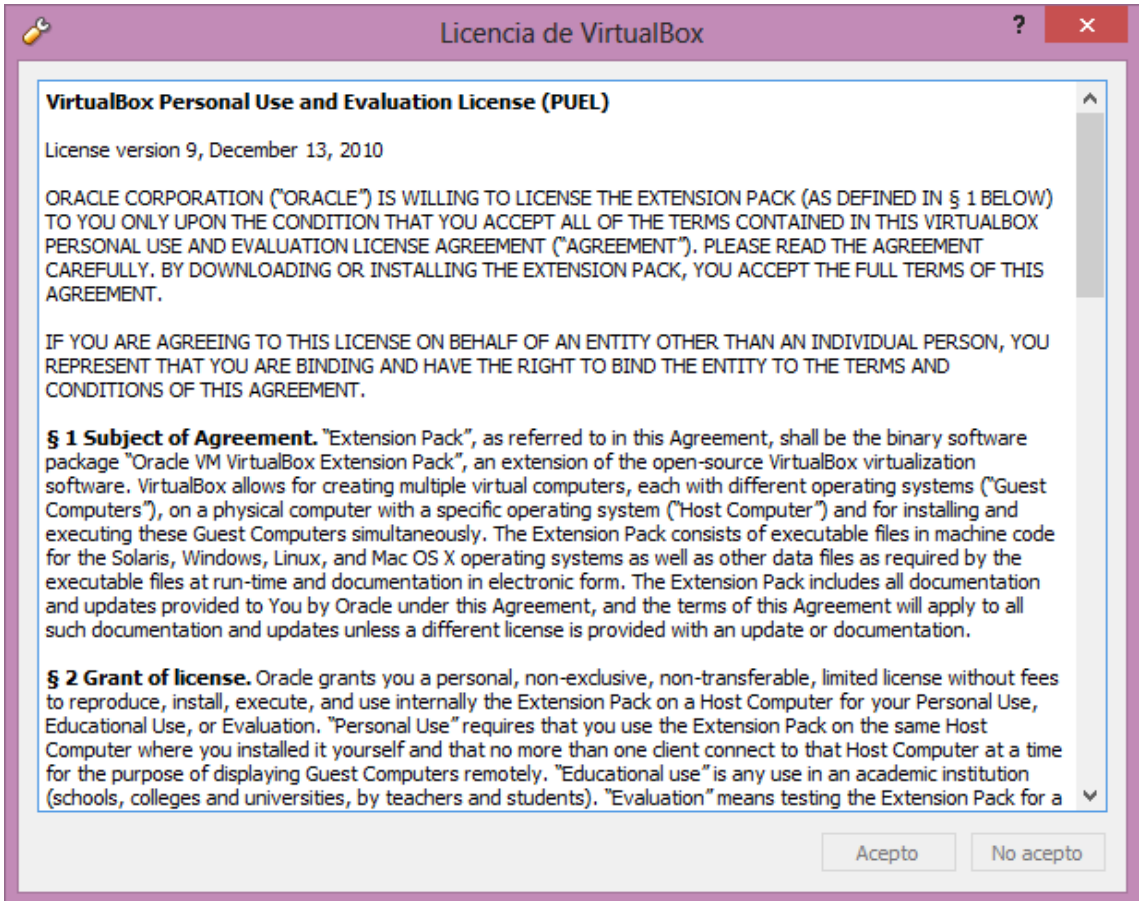


Y damos Click en el icono  y nos aparecerá una ventana donde tenemos que buscar en la ruta donde descargamos la extensión y la seleccionamos.

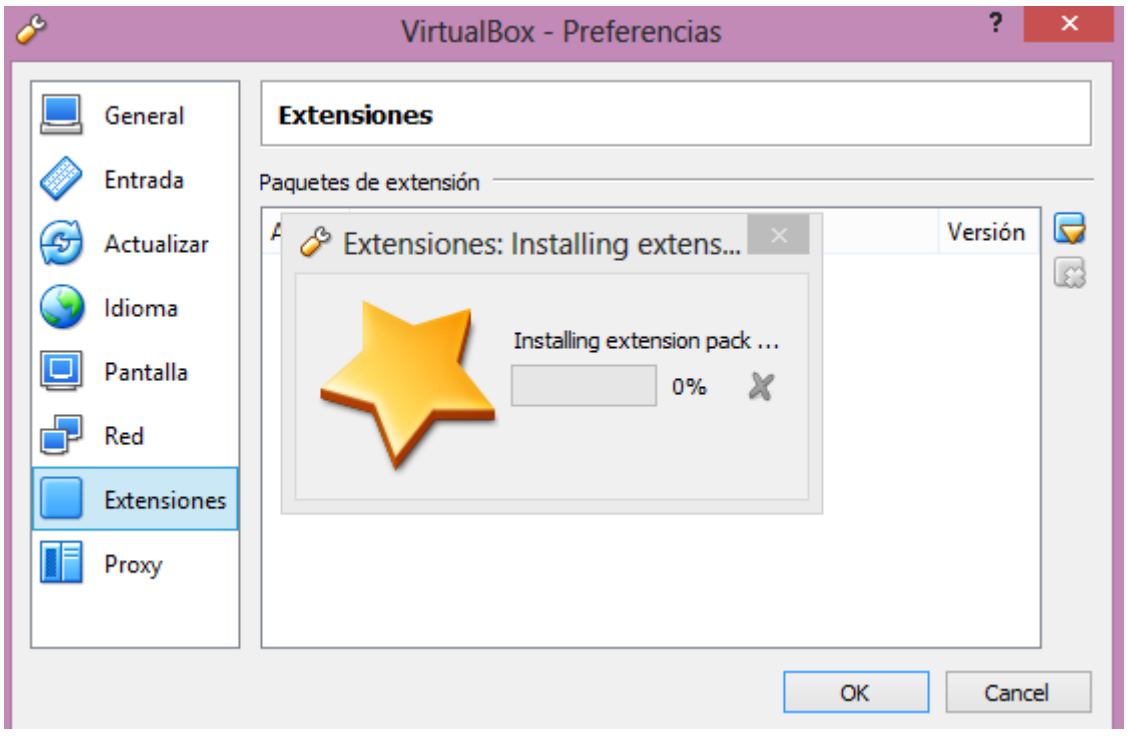
Oracle_VM_VirtualBox_Extension_Pack-5.1.2-108956

Nos aparecerá un mensaje de advertencia preguntándonos si queremos instalar dicha extensión, presionamos el botón instalar.

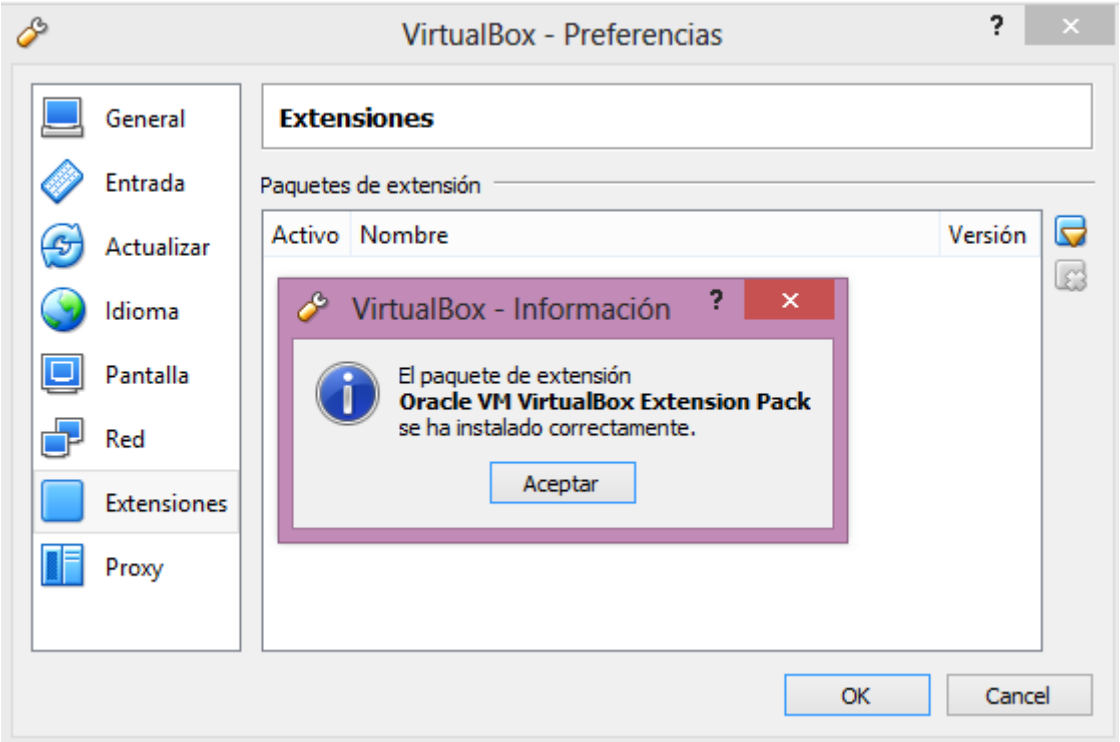
Nos aparecerán los términos de la licencia y damos nuevamente a aceptar.



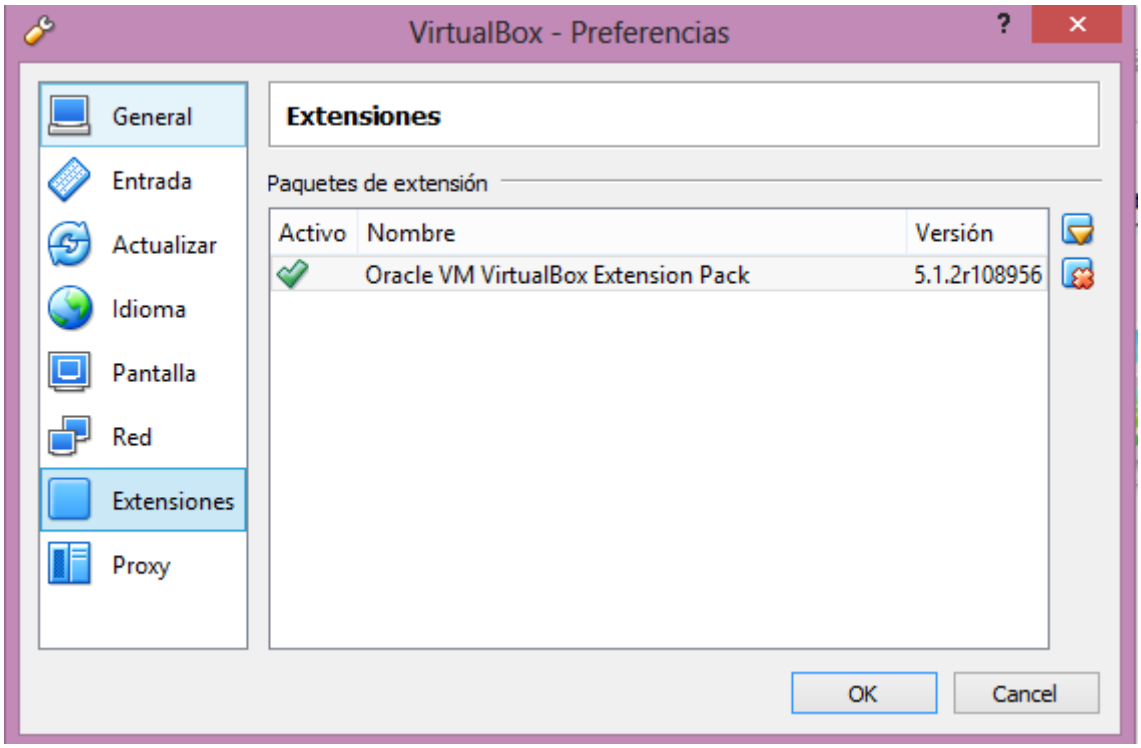
Aparecerá una nueva ventana indicándonos el estado de instalación.



Si todo sale correctamente, nos aparecerá un cuadro de dialogo que indica que la instalación fue realizada con éxito, damos aceptar.



Y comprobamos que todo quedo instalado correctamente.



Con esta imagen damos por finalizado el paso de instalación de Virtual Box.

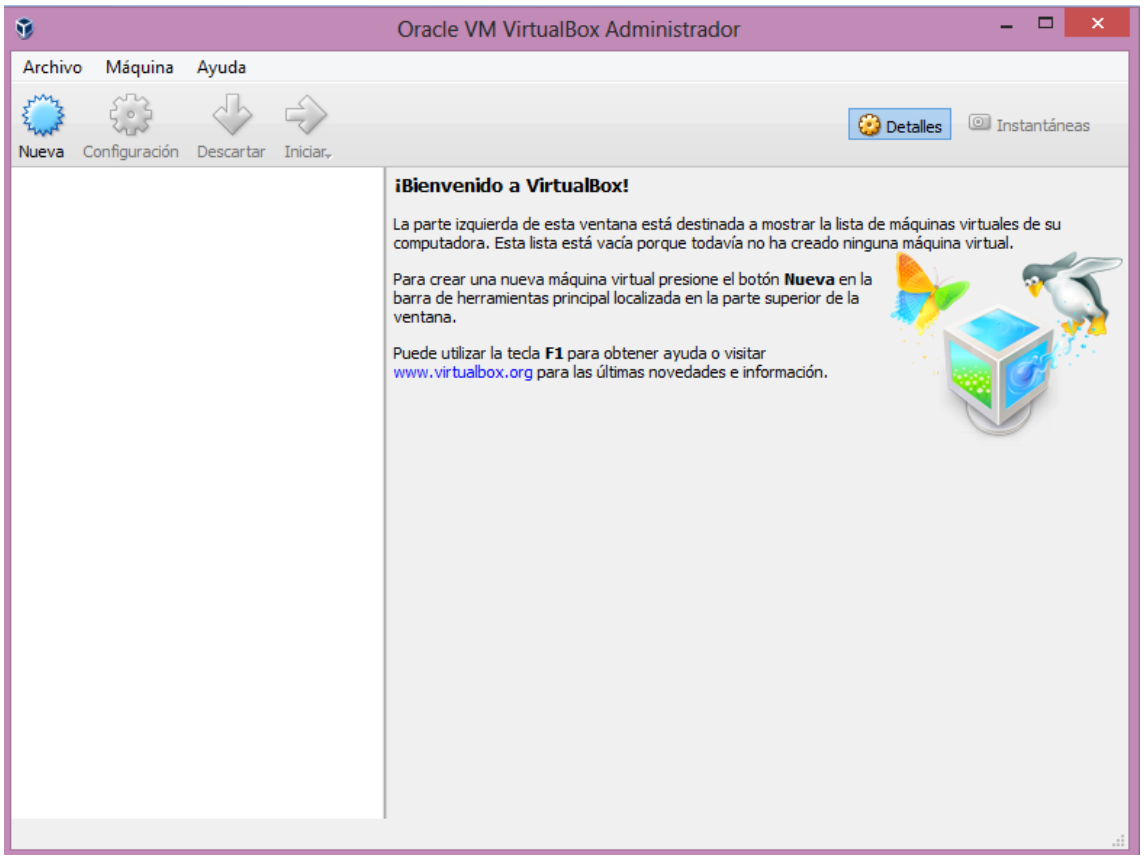
Creando nuestra maquina Virtual de Pruebas

En este paso lo primero que tenemos que hacer es realizar una descarga de nuestro sistema operativo, para esto vamos a el siguiente link en nuestro navegador.

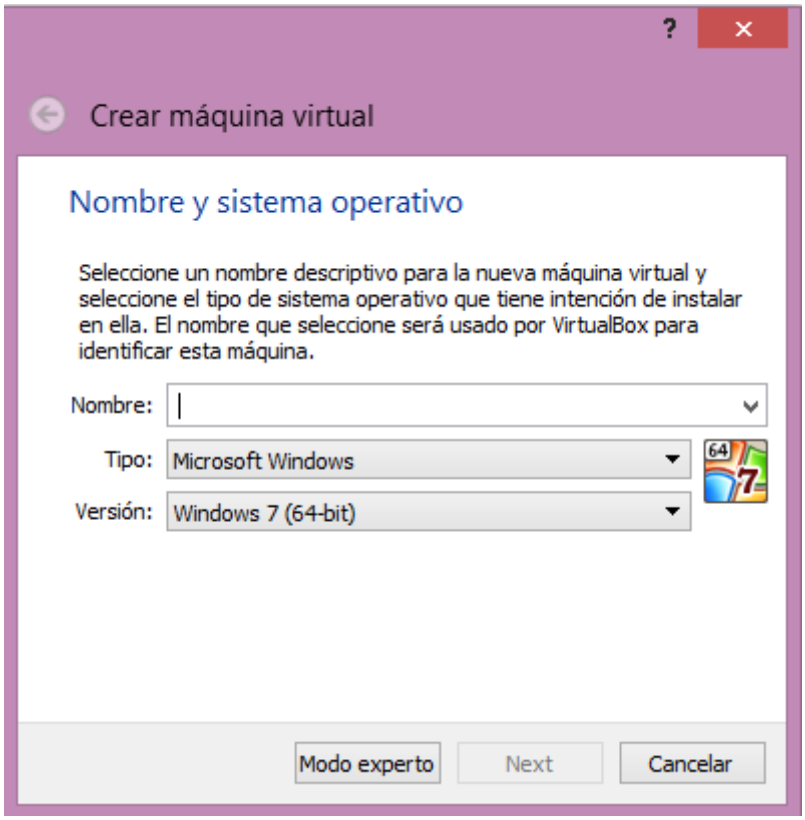
http://old-releases.ubuntu.com/releases/14.04.0/ y Descargamos la versión http://old-releases.ubuntu.com/releases/14.04.0/ubuntu-14.04.1-desktop-i386.iso

El archivo es una imagen ISO que pesa alrededor de 987 MB

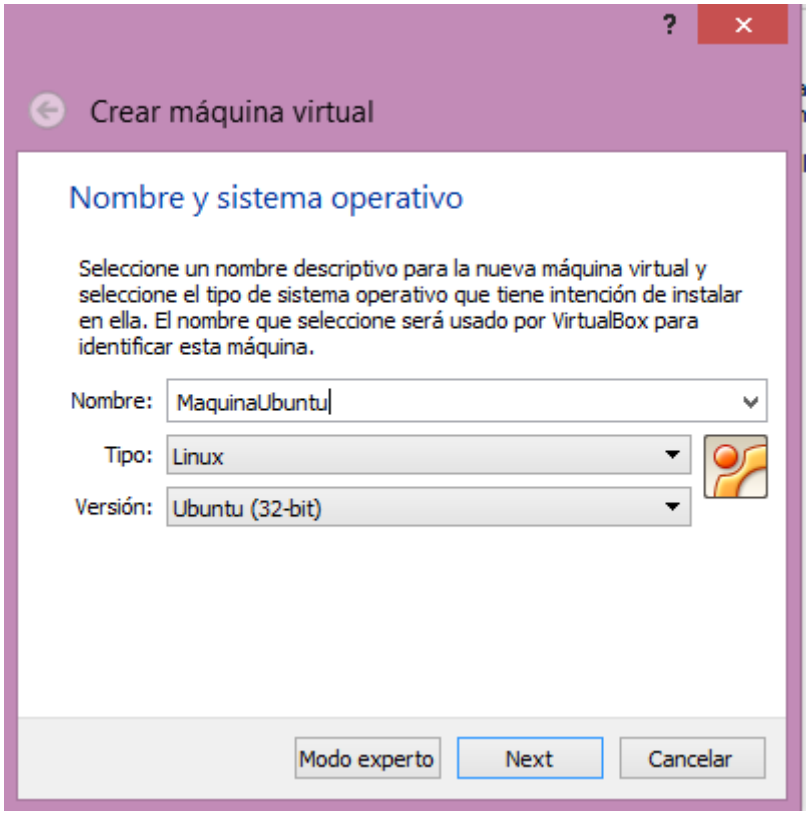
Una vez terminada la descarga, abrimos nuestro Virtual Box y damos click en el icono Nueva.



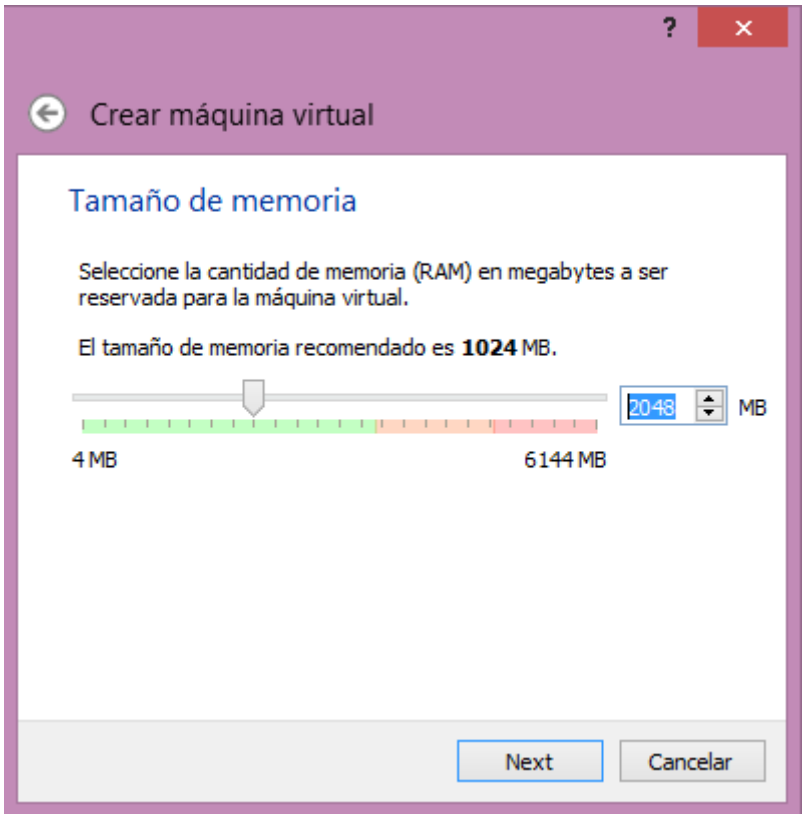
Nos aparecerá el siguiente formulario que debemos completar.



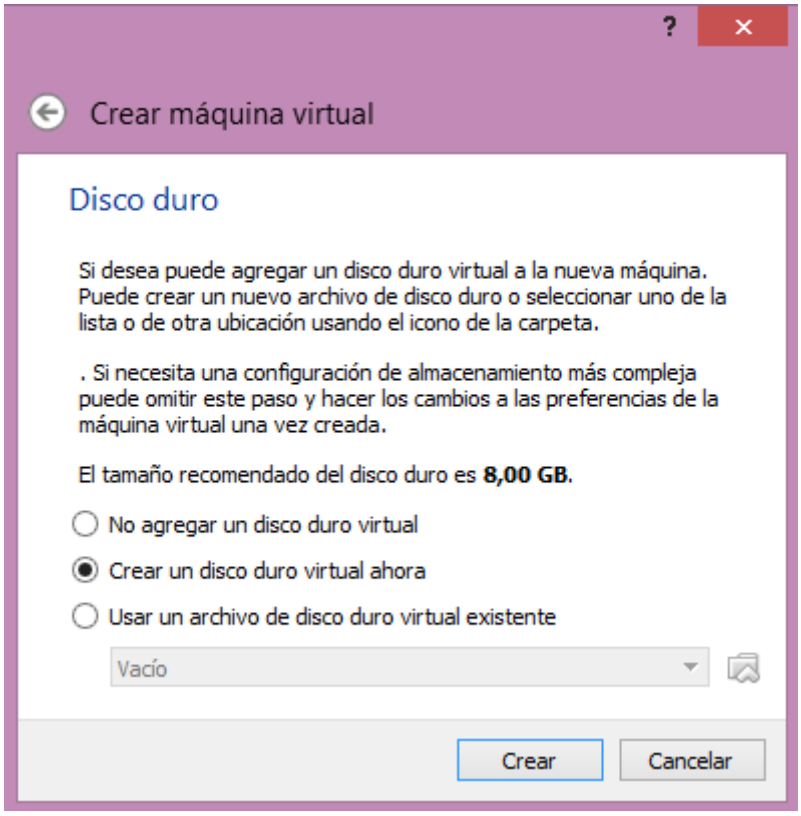
Tenemos que escoger un nombre, sistema operativo, versión y damos Next.



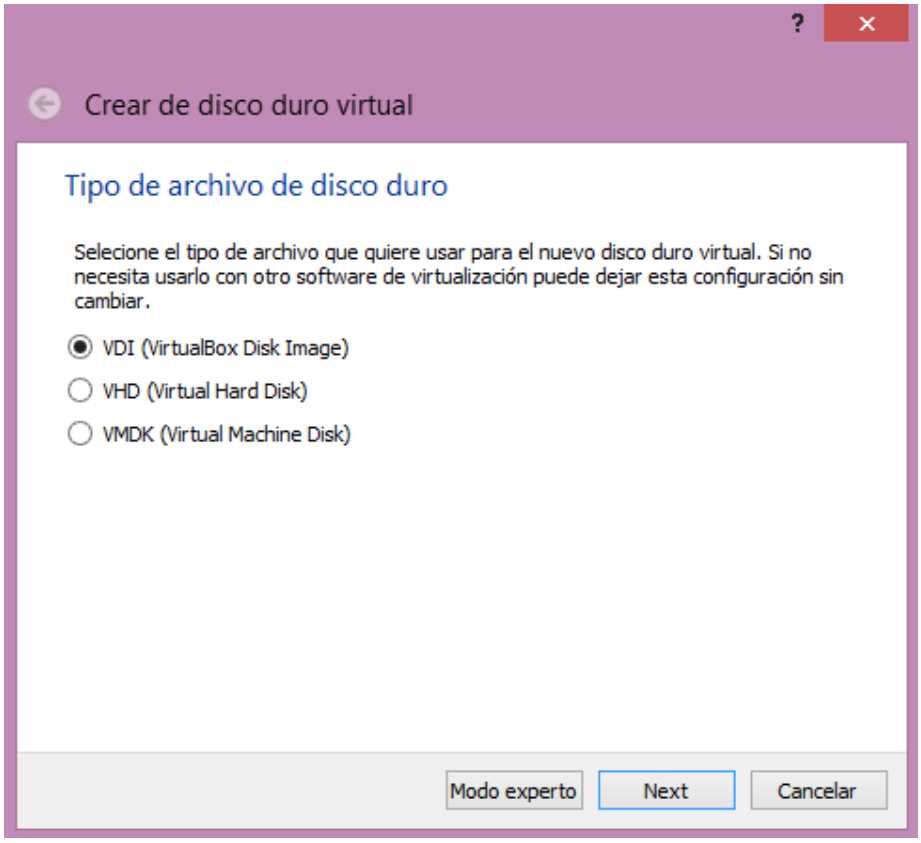
Se solicita que asignemos Memoria RAM a nuestra maquina Virtual. En el caso de Ubuntu lo recomendable es reservar por lo menos 2048 MB. Hacemos click en Next.



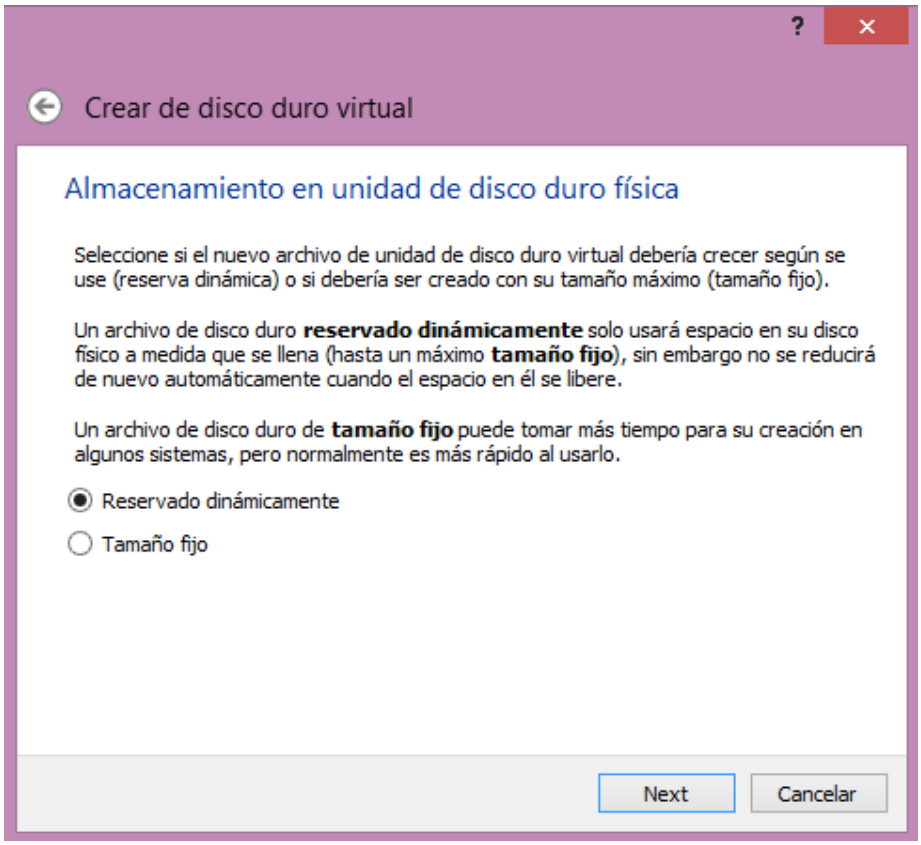
Creamos un disco duro Virtual y apretamos el botón Crear.



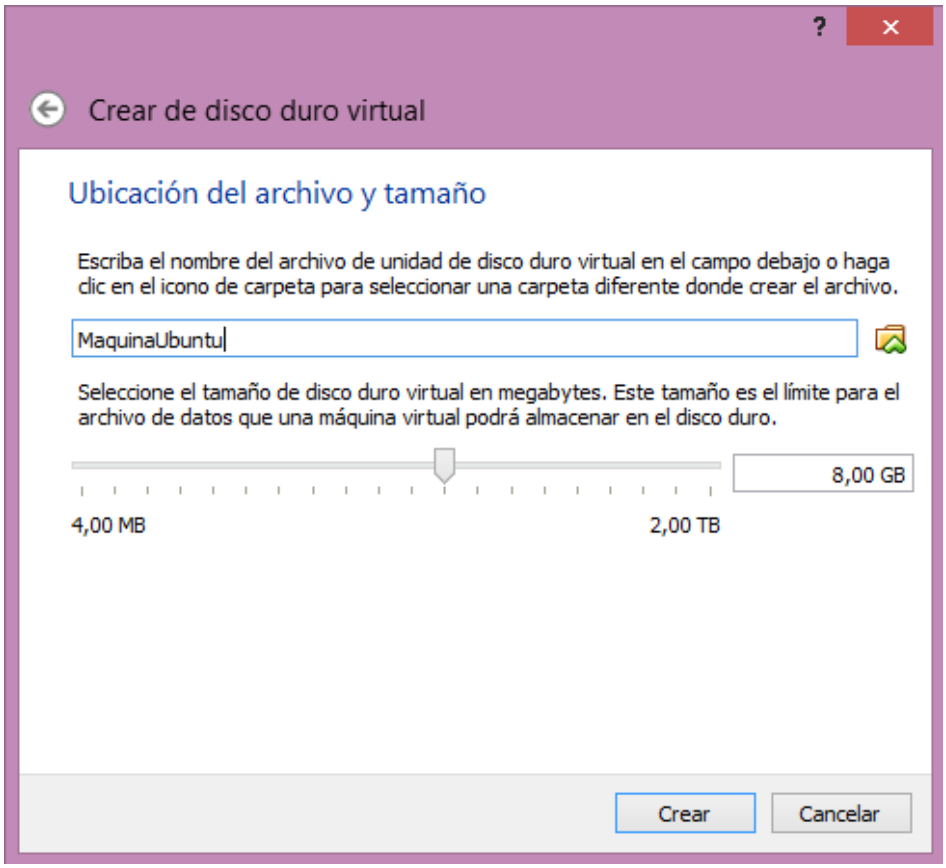
Dejamos las opciones por defecto y damos Next.



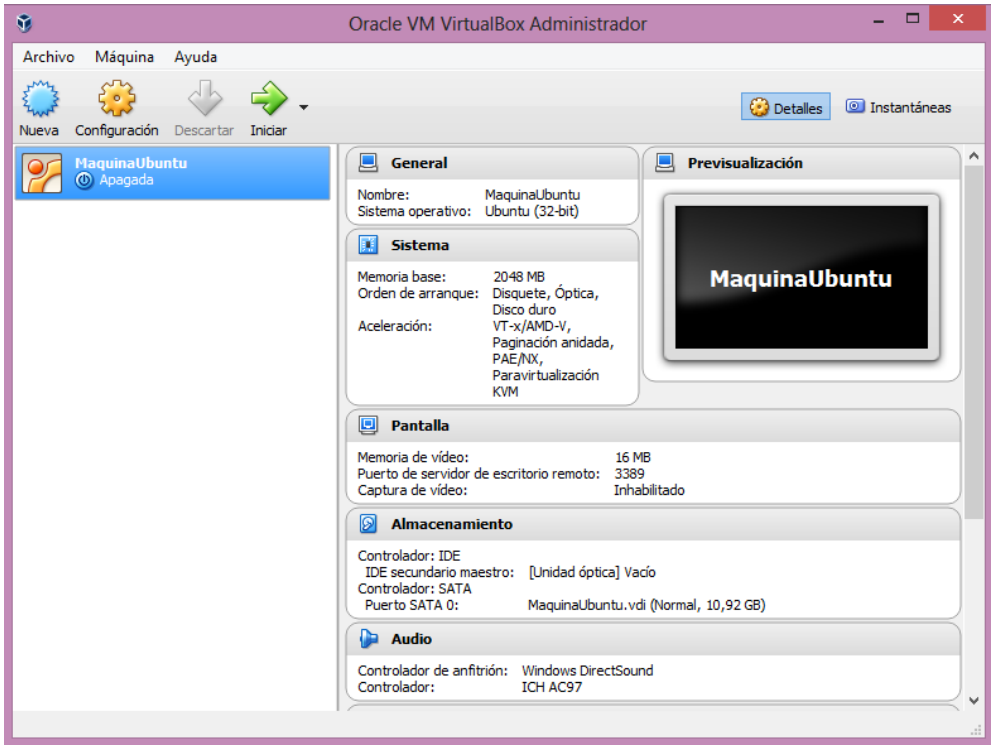
Nuevamente Hacemos Click en Next.



Seleccionamos el espacio de memoria que vamos a reservar para que sea usado por nuestro disco duro Virtual. Damos click a el botón Crear.

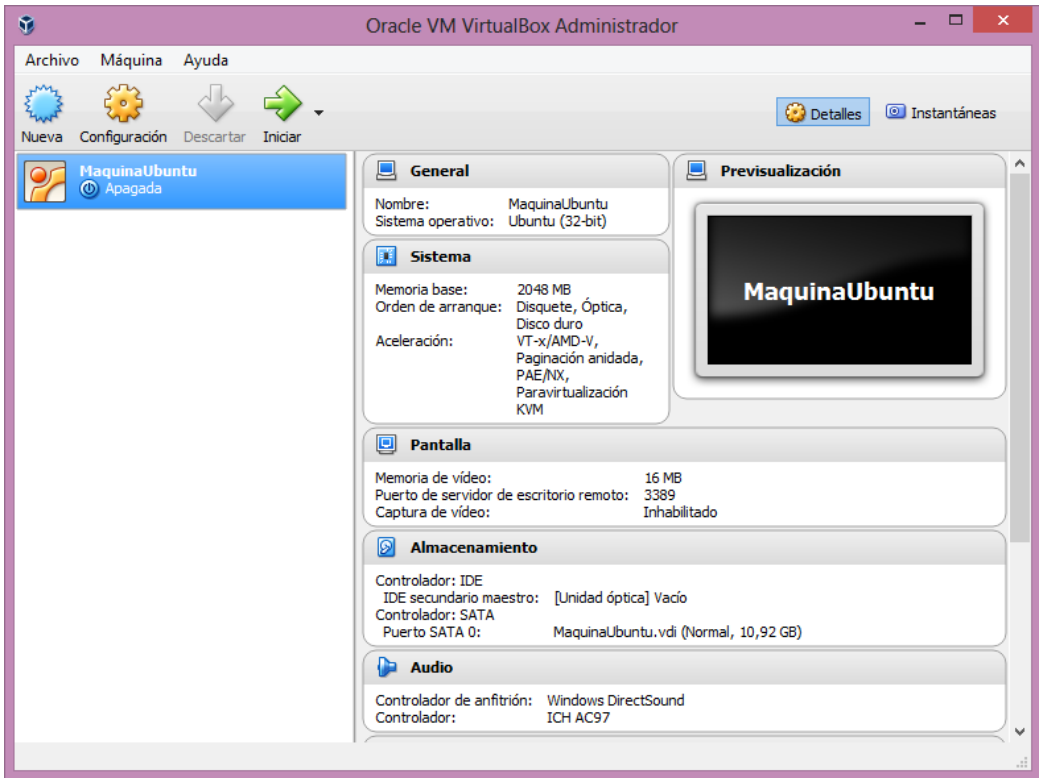


Si todo salio bien, nos aparecerá la siguiente Imagen.

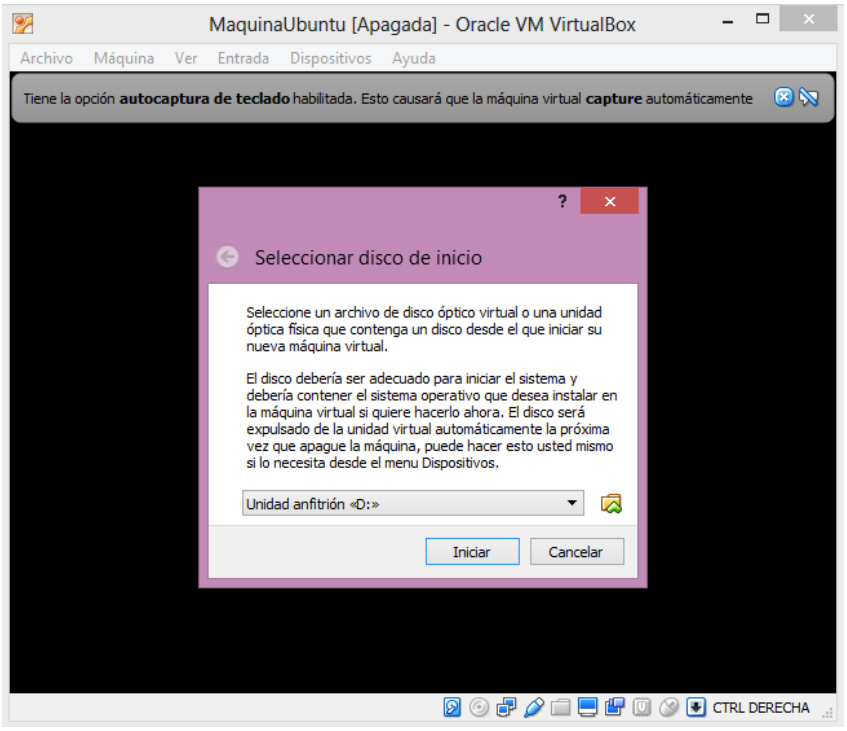


Configurando Nuestro Ubuntu

Para comenzar con la configuración de nuestro Ubuntu vamos a dar click sobre el botón Iniciar.

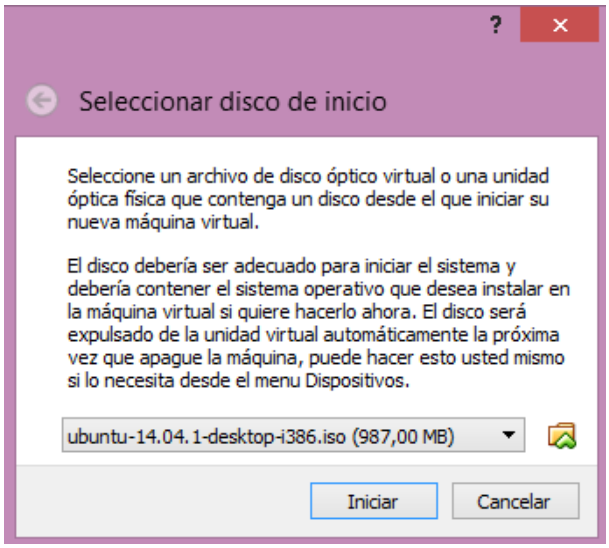


Virtual Box desplegara una segunda Ventana donde nos pedirá seleccionar la imagen del sistema operativo, en nuestro caso seleccionamos la ISO que descargamos anteriormente.

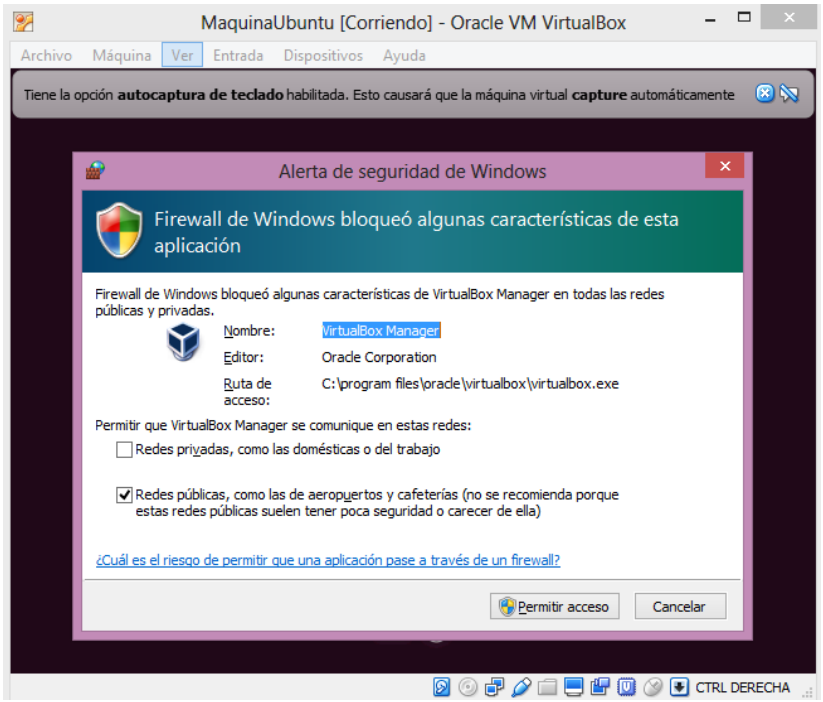


Damos click sobre el icono  y seleccionamos la ISO de ubuntu.

Una vez seleccionada la imagen, hacemos click en Iniciar.



La primera vez nos la instalación nos pedirá permiso para no ser bloqueado por el Firewall.

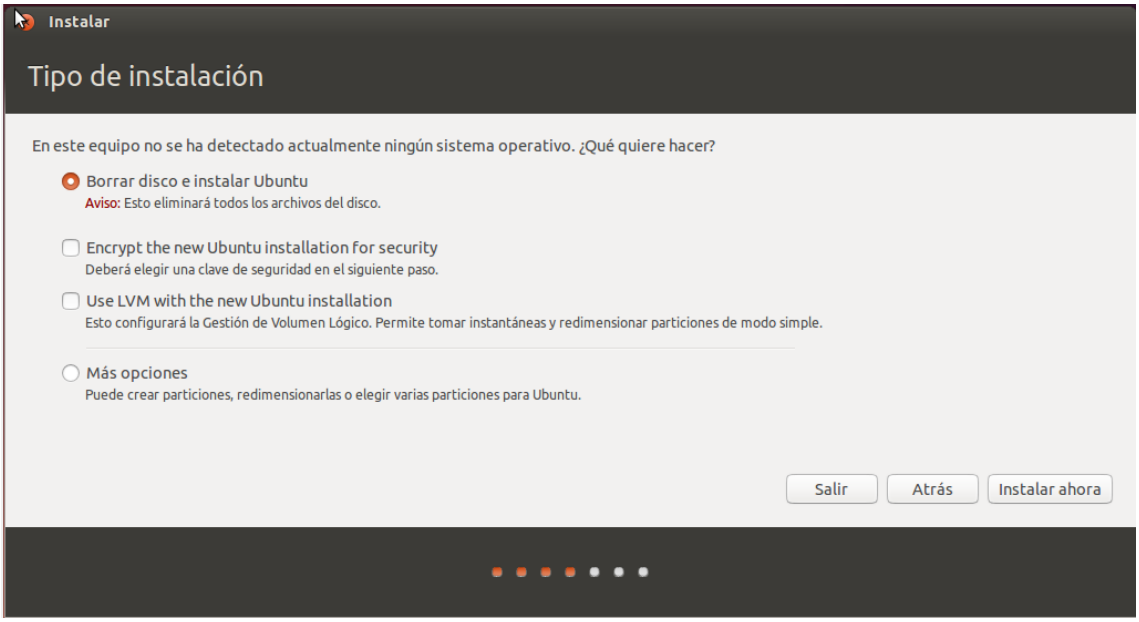


Seleccionamos el idioma que deseamos en nuestro Sistema Operativo y damos click sobre Instalar Ubuntu.

.

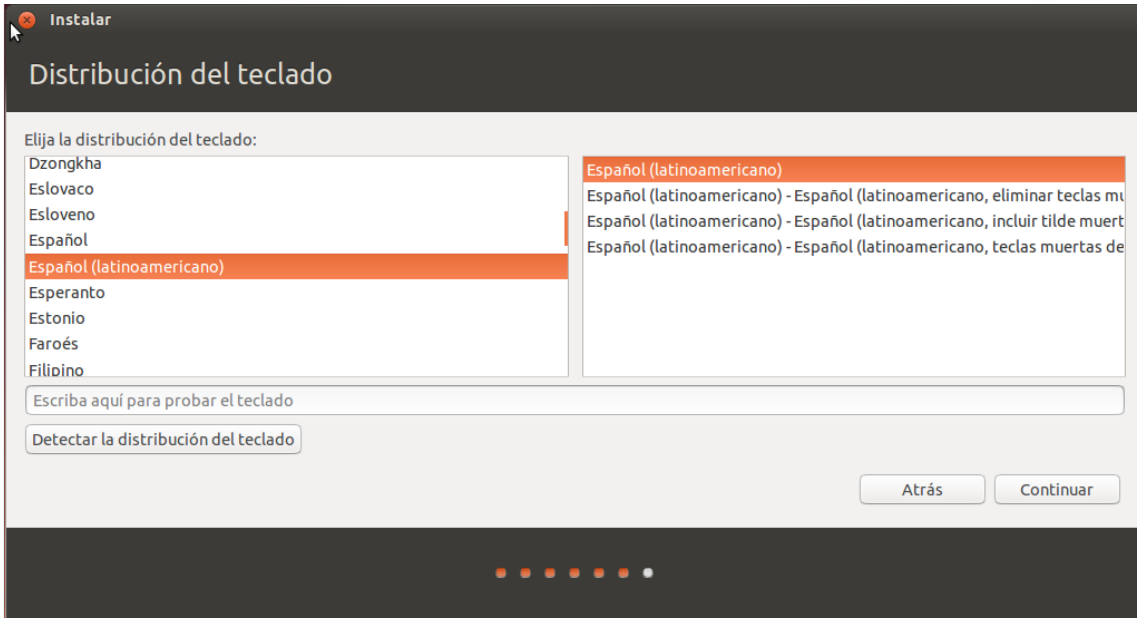
Ubuntu nos solicita ciertos requisitos antes de instalar y damos Click a continuar.

Damos click a instalar ahora.

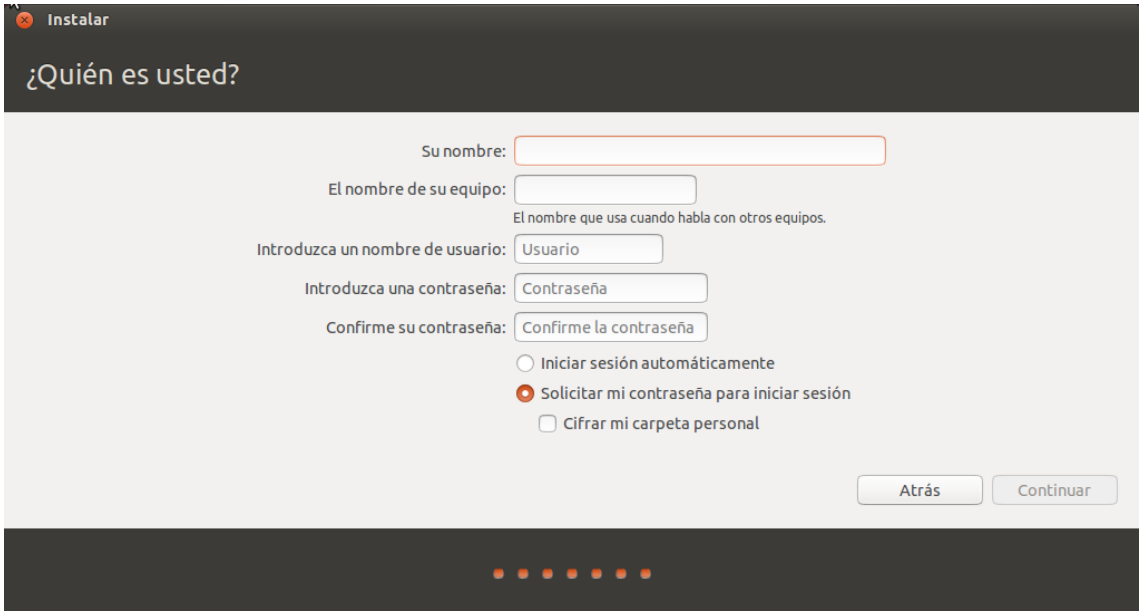


Seleccionamos nuestra ubicación.

Seleccionamos la configuración del teclado.

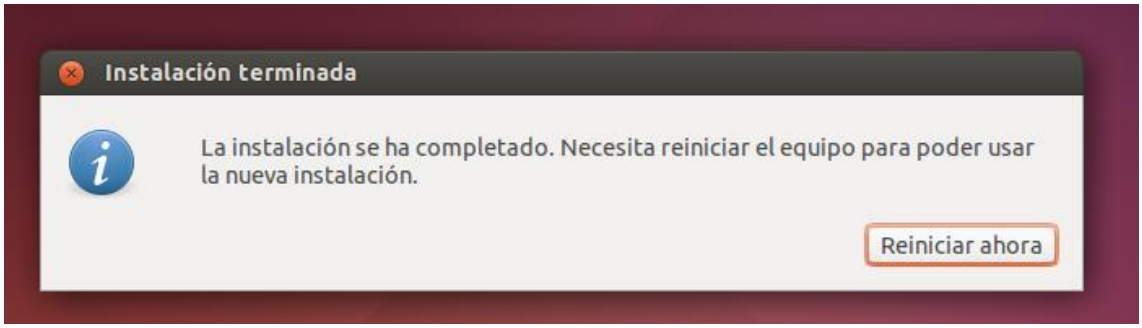


Completamos los campos solicitados.



Si todo sale bien nos aparecerá la siguiente pantalla, lo que indica que tenemos que esperar.

Una vez finalizada la instalación, nos aparecerá el siguiente cuadro de dialogo.



Damos click en reiniciar ahora y con esto terminamos la instalación.

Instalando nuestro primer IDE para crear nuestro primer programa.

Antes de la instalación, me gustaría detenerme a explicar que comandos debemos utilizar en nuestro GNU/Linux para instalar cualquier aplicación.

sudo: El programa sudo (del inglés super user do) es una utilidad de los sistemas operativos tipo Unix, como Linux, BSD, o Mac OS X, que permite a los usuarios ejecutar programas con los privilegios de seguridad de otro usuario (normalmente el usuario root) de manera segura, convirtiéndose así temporalmente en súper usuario.

apt-get: El sistema de paquetes utiliza una base de datos para llevar un monitoreo de los paquetes instalados, los no instalados y cuales están disponibles para su futura instalación. El programa apt-get utiliza esta base de datos para averiguar como instalar los paquetes que son requeridos por el usuario y para indagar sobre que paquetes adicionales serán requeridos para que el paquete seleccionado funcione correctamente.

apt-get update: Actualiza la lista de paquetes disponibles y sus versiones, pero no instala o actualiza ningún paquete. Esta lista la coge de los servidores con repositorios que tenemos definidos en el /etc/apt/sources.list.

apt-get upgrade: una vez el comando anterior ha descargado la lista de software disponible y la versión en la que se encuentra, podemos actualizar dichos paquetes usando este comando: apt-get upgrade. Instalará las nuevas versiones respetando la configuración del software cuando sea posible.

Ya conociendo los comandos a utilizar procedemos a instalar nuestro IDE con la siguiente instrucción en nuestra Terminal de Ubuntu:

```
alumno@hackerpython:~$ sudo apt-get install ninja-ide
```

Una vez ejecutada esta instrucción nos aparecerán los siguientes mensajes en pantalla:

```
alumno@hackerpython:~$ sudo apt-get install ninja-ide
[sudo] password for alumno:
```

Nos solicitara la contraseña del usuario sudo, la escribimos en la Terminal y damos entrar.

```
Leyendo lista de paquetes... Hecho
Creando árbol de dependencias
Leyendo la información de estado... Hecho
```

Se leen en las dependencias si existe un programa llamado ninja-ide, si es correcto, se buscan los programas a descargar, se verifican si el programa depende de otros programas para funcionar y se crean índices de dependencias.

Se define que programas nuevos se van a instalar y los paquetes extras que se necesitan. Luego de esto se descargan en nuestra maquina.

```
Se instalarán los siguientes paquetes extras:
  python-pyinotify
Paquetes sugeridos:
  python-pyinotify-doc
Se instalarán los siguientes paquetes NUEVOS:
  ninja-ide python-pyinotify
0 actualizados, 2 se instalarán, 0 para eliminar y 610 no actualizados.
Necesito descargar 568 kB de archivos.
Se utilizarán 2.043 kB de espacio de disco adicional después de esta operación.
¿Desea continuar? [S/n] s
Des:1 http://cl.archive.ubuntu.com/ubuntu/ trusty/main python-pyinotify all 0.9.4
-1build1 [24,5 kB]
Des:2 http://cl.archive.ubuntu.com/ubuntu/ trusty/universe ninja-ide all 2.3-2 [5
43 kB]
Descargados 568 kB en 2seg. (259 kB/s)
Seleccionando el paquete python-pyinotify previamente no seleccionado.
(Leyendo la base de datos ... 165969 ficheros o directorios instalados actualment
e.)
```

Se desempaquetan los programas y se proceden a instalar.

```
Preparando para desempaquetar .../python-pyinotify_0.9.4-1build1_all.deb ...
Desempaquetando python-pyinotify (0.9.4-1build1) ...
Seleccionando el paquete ninja-ide previamente no seleccionado.
Preparando para desempaquetar .../ninja-ide_2.3-2_all.deb ...
Desempaquetando ninja-ide (2.3-2) ...
Procesando disparadores para man-db (2.6.7.1-1) ...
Procesando disparadores para gnome-menus (3.10.1-0ubuntu2) ...
Procesando disparadores para desktop-file-utils (0.22-1ubuntu1) ...
Procesando disparadores para bamfdaemon (0.5.1+14.04.20140409-0ubuntu1) ...
Rebuilding /usr/share/applications/bamf-2.index...
Procesando disparadores para mime-support (3.54ubuntu1) ...
Configurando python-pyinotify (0.9.4-1build1) ...
Configurando ninja-ide (2.3-2) ...
alumno@hackerpython:~$
```

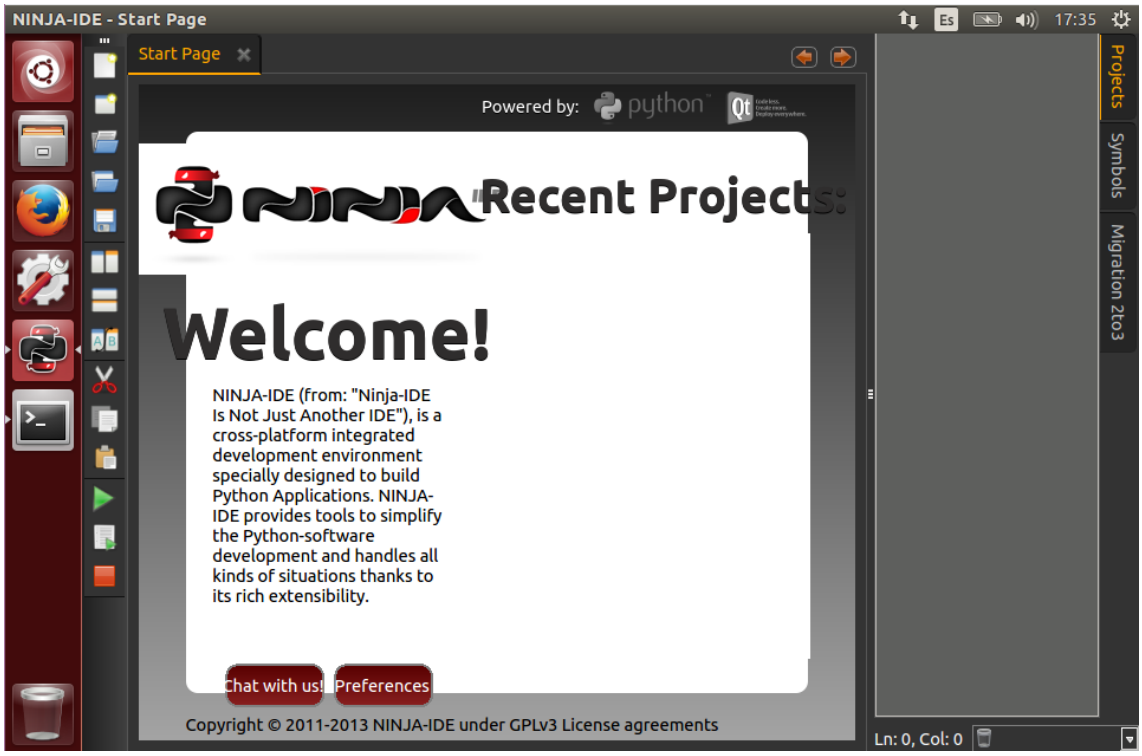
Con estos pasos ya tenemos nuestro IDE, descargado y funcionando.

IDE: Es un entorno de desarrollo integrado o entorno de desarrollo interactivo, en inglés Integrated Development Environment (IDE), es una aplicación informática que proporciona servicios integrales para facilitarle al desarrollador o programador el desarrollo de software.

Para acceder a nuestro programa escribimos en la Terminal el siguiente comando:

```
alumno@hackerpython:~$ ninja-ide
```

Y se nos abrirá la siguiente interfaz grafica:



Uso del intérprete en Python

02

Comenzando a programar

Para dar los primeros pasos en la programación de computadoras, tenemos que comprender la filosofía del lenguaje y como tenemos que transmitir nuestros pensamientos a la computadora. Cuando comenzamos a realizar un programa, debemos tomar en cuenta que nuestra maquina solo entiende operaciones matemáticas y comparaciones (verdadero o falso). Por lo tanto nuestros pensamientos tienen que ser acorde a lo que las maquinas pueden hacer por nosotros y dividir nuestras mentes para hablar lenguaje maquina y humano a la vez. A mi me gusta pensar que nuestra misión como programador, es enseñarle a la computadora a pensar y hacer la vida mas fácil a las personas. Cuando un programa se ejecuta, es el pensamiento de un humano escrito en esa maquina, cuando pensamos que una maquina es inteligente, estamos alabando a uno a varios programadores que hicieron una tarea imposible.

Cuando comenzamos a programar, estamos relatando el como nosotros nos imaginamos que se realiza una tarea, es decir, es muy parecido a escribir un libro o una receta de cocina. La diferencia que nosotros aparte de escribir el conjunto de tareas a realizar, tenemos que pensar que otras cosas pueden suceder si este programa se ejecuta. Un buen ejemplo seria escribir un programa que ataque un servidor Web, si nos concentramos solamente en que el servidor HTTP esta en el puerto 80 y lo dejamos declarado en el código y además no damos la posibilidad de cambiarlo, estaríamos cometiendo un error de programación, por que un servicio Web puede estar abierto en cualquier puerto y el programa solo funcionaria en el puerto por defecto, estaríamos perdiendo la posibilidad de auditar otras aplicaciones, que pudiesen ser vulnerables a algún tipo de ataque.

En Python tenemos dos formas de realizar un programa, la primera es mediante la consola interactiva y la segunda escribiendo en un archivo con extensión .py.

La consola interactiva o intérprete: sirve para escribir tareas breves y rápidas, requiere un dominio muy bueno del lenguaje, ya que cada instrucción enviada es ejecutada inmediatamente. Personalmente la uso poco, la mayor parte de programas que realizo son archivos perdurables.

- Ejecución de script o archivos .py: Esta forma de hacer programas en Python es la más utilizada, puedes guardar tu código fuente, compartirlo o modificarlo cuando quieras.

- Las dos formas son buenas y sirven para cada necesidad en especial, para rapidez se recomienda uso de consola, para programas perdurables en el tiempo recomiendo archivos .py

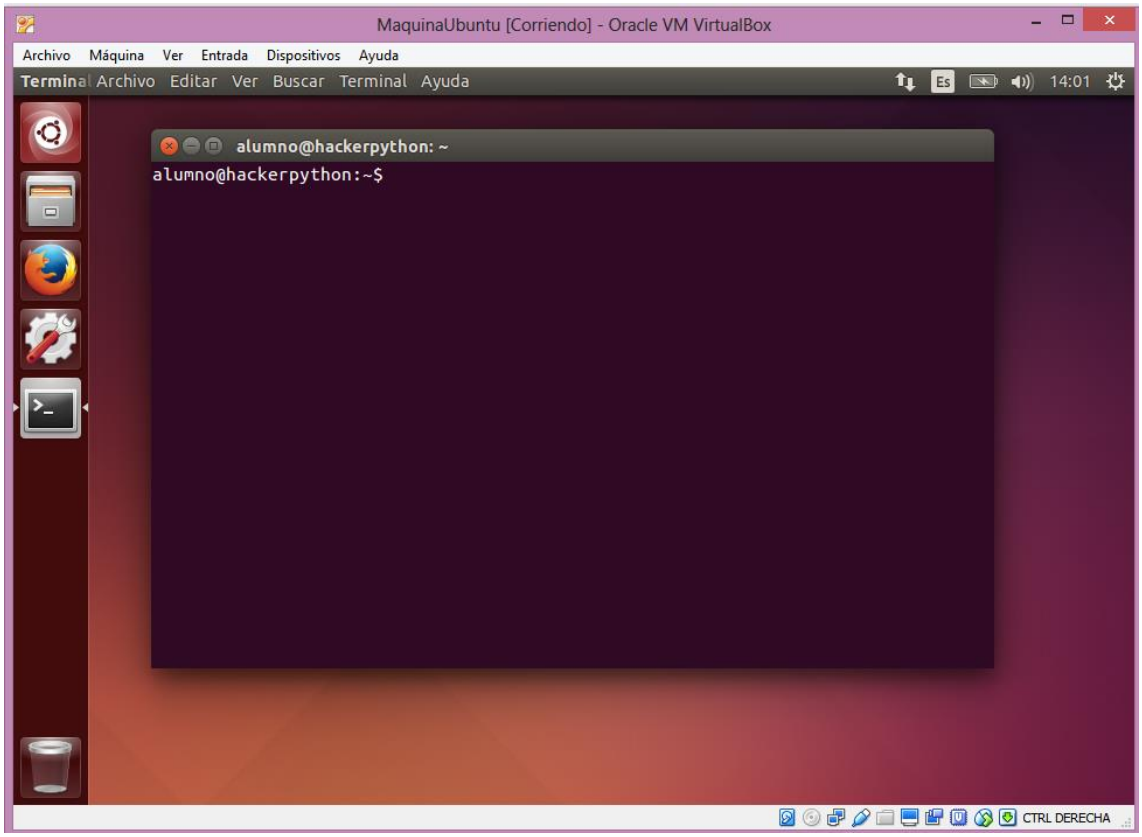
Uso del intérprete de Python

La mejor forma de iniciar con Python es hacer uso de la consola interactiva, jugaremos un poco con ella, antes de escribir nuestros programas.

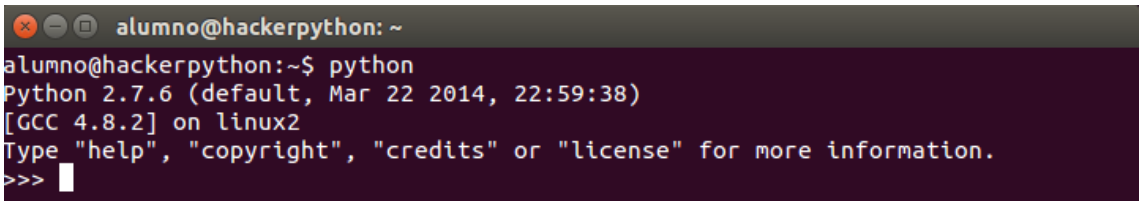
Por lo general python suele estar instalado en la siguiente ruta

```
/usr/local/bin/python
```

Para iniciar con la consola interactiva, vamos a nuestra maquina Ubuntu y vamos a escribir el atajo por teclado: Ctrl + Alt + T



Teniendo la consola abierta, vamos a escribir la palabra **python** en ella y damos Entrar. Esta ruta **/usr/local/bin/python** hace posible que el intérprete de Python se ejecute con este comando.



El símbolo >>> indica que la consola interactiva esta lista para ser usada.

El primer ejemplo que vamos a hacer es escribir:

```
>>> ip = "192.168.1.33"
```

Lo que acabamos de hacer es indicarle a la consola que vamos a guardar 192.168.1.33 en la variable ip. Cada vez que queramos saber que valor tiene guardada la variable ip escribiremos:

```
>>> ip
'192.168.1.33'
>>>
```

Para asignar un valor a una variable se debe utilizar el símbolo =
A las variables se le puede asignar el nombre que nosotros queramos, en este caso se escogió ip, pero puede ser cualquier nombre.

Si queremos sumar dos números en la consola, se debe ingresar de la siguiente manera:

```
>>> 16+17
33
>>>
```

Como se puede apreciar, la consola sumo inmediatamente los dos números. Por otra parte si queremos restar, realizamos la siguiente operación:

```
>>> 18-15
3
>>>
```

Si queremos sumar dos números asignados a variables, se debe hacer de la siguiente manera:

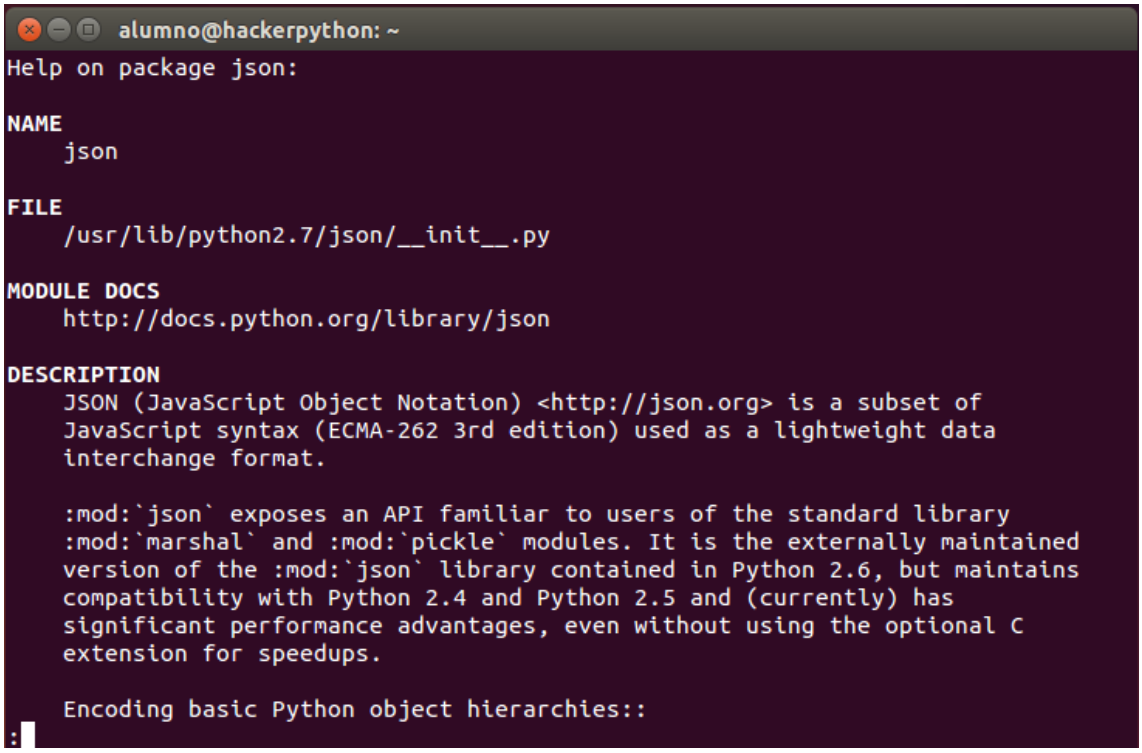
```
>>> a = 16
>>> b = 17
>>> a+b
33
>>>
```

Si se desea saber sobre el uso de una librería en especial, deberíamos utilizar la función help () de la siguiente manera.

```
>>> help('json')
```

El resultado de esto, debería contener la descripción de la librería y como se utiliza.

Resultado de la función help ()



Debemos salir de la descripción ingresando la letra q.

Si queremos saber si contamos con alguna librería en especial como por ejemplo MySQL, que sabemos que no viene instalado por defecto en Python, escribimos:

```
>>> help('MySQLdb')
no Python documentation found for 'MySQLdb'

>>>
```

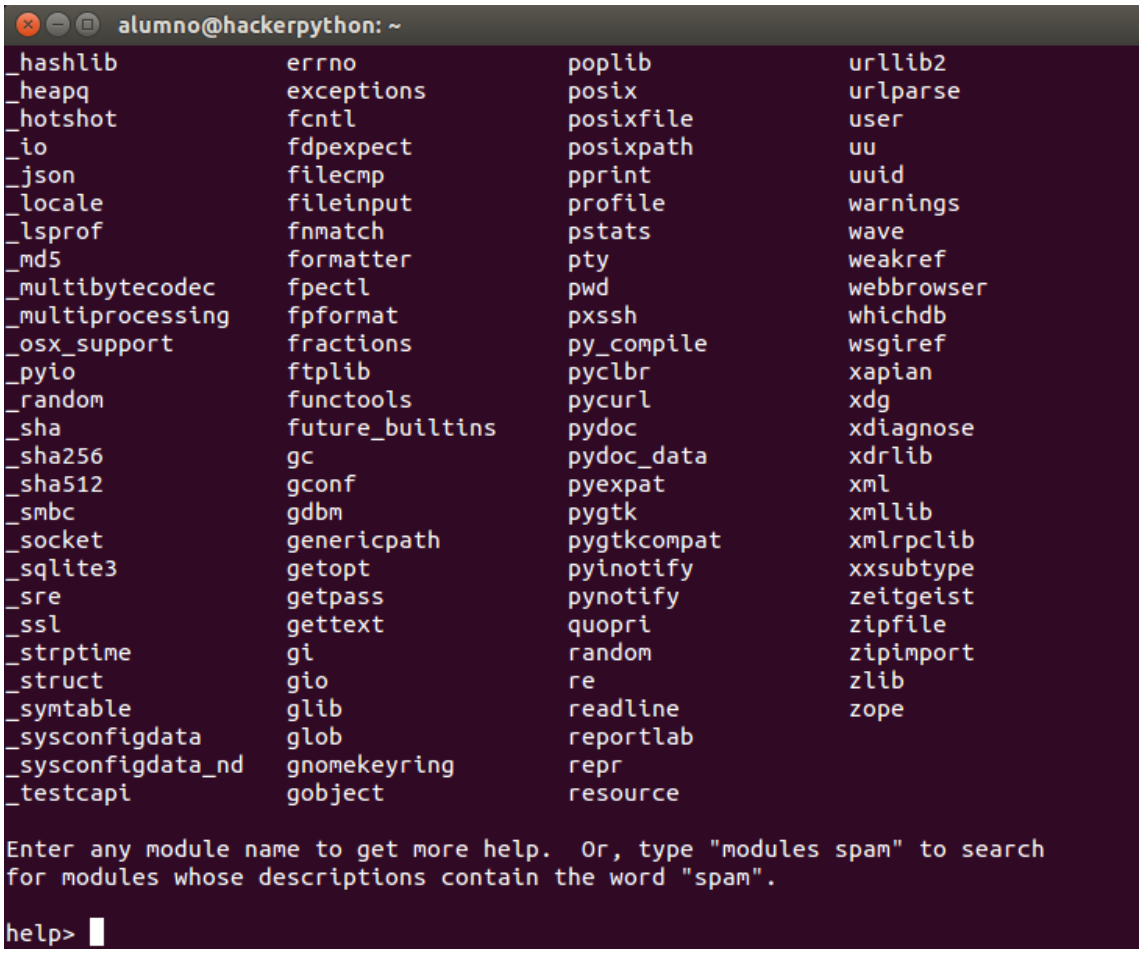
Con el comando help () escrito sin ningún pasarle ningún parámetro, le podemos pedir ayuda al intérprete:

```
>>> help()

Welcome to Python 2.7!  This is the online help utility.

If this is your first time using Python, you should definitely check out
the tutorial on the Internet at http://docs.python.org/2.7/tutorial/.

Enter the name of any module, keyword, or topic to get help on writing
Python programs and using Python modules.  To quit this help utility and
return to the interpreter, just type "quit".

To get a list of available modules, keywords, or topics, type "modules",
"keywords", or "topics".  Each module also comes with a one-line summary
of what it does; to list the modules whose summaries contain a given word
such as "spam", type "modules spam".

help>
```

Por ejemplo, si seguimos con el ejemplo anterior le podemos preguntar al intérprete que módulos tiene instalado, con el comando modules:

```
help> modules

Please wait a moment while I gather a list of all available modules...
```

Y nos entregara la lista:



Con el comando quit, podemos salir de la ayuda del interprete

```
help> quit
```

Con el comando exit(), podemos salir del interprete:

```
>>> exit()
```

Elementos Del Lenguaje

03

Uso de Variables y Constantes

Muchas veces cuando estamos escribiendo un programa, necesitamos almacenar valores para su uso posterior, estos valores podrían ser, de tipo texto, numéricos, boléanos, alfanuméricos, etc. Si los valores que estamos guardando no van a cambiar en el transcurso del programa, las podemos definir como constantes. En caso contrario, sino estamos seguros del tratamiento que le vamos a dar a nuestras variables o los valores almacenados van a cambiar a través del tiempo, las podemos definir variables.

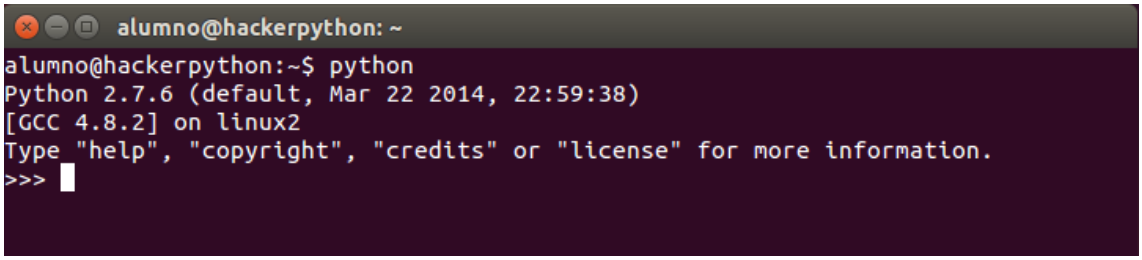
La forma de definir una variable es la siguiente:

```
mi_variable = contenido_asignado_a_la_variable
```

La forma de definir una constante es:

```
MI_CONSTANTE  = contenido_asignado_a_la_variable
```

Para explicar mejor este tema vamos a usar la consola interactiva de python.



El siguiente ejemplo muestra la forma correcta de definir una variable:

```
>>> mi_variable = "contenido guardado"
>>>
```

La forma incorrecta de definir una variable seria la siguiente:

Al utilizar espacios al definir una variable, el intérprete de python no lo procesaría.

```
>>> Mi variable = "otro contenido"
  File "<stdin>", line 1
    Mi variable = "otro contenido"
               ^
SyntaxError: invalid syntax
>>>
```

Al utilizar caracteres como por ejemplo $ %&/()=?¿!

```
>>> $variable = "otro contenido"
  File "<stdin>", line 1
    $variable = "otro contenido"
    ^
SyntaxError: invalid syntax
>>>
```

Al comenzar una definición de variables con números.

```
>>> 1variable = "otro contenido"
  File "<stdin>", line 1
    1variable = "otro contenido"
             ^
SyntaxError: invalid syntax
```

Para trabajar con variables se debe tomar en cuenta que se deben utilizar nombres descriptivos, si se quiere utilizar mas de una palabra se debe escribir con guión bajo y finalmente cabe destacar que, la sintaxis correcta para utilizar el signo '=' es fijarse que siempre respetemos el utilizar un solo espacio después del signo igual.

Para definir una constante es de la siguiente forma:

```
>>> MI_CONSTANTE = 3,1416
>>>
```

Como se observa en la captura de pantalla que se utilizan letras mayúsculas y además se debe utilizar guión bajo para separar las palabras, las reglas son parecidas a las variables para no tener errores de sintaxis.

Para imprimir el valor de una variable o constante en pantalla se utiliza la palabra reservada **print**

```
>>> print MI_CONSTANTE
3.1416
>>>
```

Tipos de Datos

04

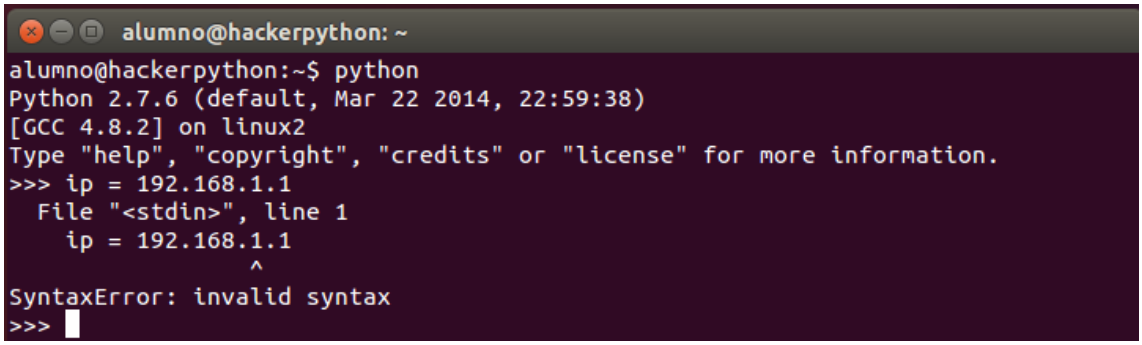
Tipos de Datos

Para hablar de como se trabaja con los datos en un lenguaje de programación, tenemos que tener en claro, que tipo de información vamos a manipular desde nuestro programa. La información básicamente en una computadora se presenta como números, cadenas de texto y valores boléanos (Verdadero o Falso). Estos datos son los pilares en los que podremos presentar la información y trabajar con ella.

Para poner en contexto lo mencionado anteriormente, pondremos un caso donde se nos encarga hacer un programa que se conecte a Internet. La primera pregunta que nos tenemos que hacer es:
¿Que tipos de datos nos podremos encontrar al hacer un script que se conecte a Internet?
Respuesta: Para conectarse a Internet se necesita una IP, un puerto a la escucha y saber si la conexión fue exitosa (Verdadero o falso).

- IP es una cadena de texto Ejemplo: “192.168.1.1”
- Puerto es un número Ejemplo: 443
- Conexión exitosa Ejemplo: TRUE o FALSE

Ahora este mismo ejemplo lo pasaremos a lenguaje Python. Para esto abriremos la consola interactiva y digitaremos.



Aprovecho este error para recalcar el manejo de variables. Si analizamos lo escrito, nos damos cuenta que en la variable ip, el valor asignado son números y puntos, si solo fuesen números los que estamos digitando estaría correcto el ejemplo, pero como tenemos el carácter punto, tenemos que escribir todo entre comillas y manejarlo como cadena de texto.

El ejemplo correcto quedaría así:

Ahora no nos salio ningún error de sintaxis. Cada vez que trabajemos con cadenas de texto, tenemos que escoger si trabajamos con comillas simples o comillas dobles. Es decir si abro con comilla simple, debo cerrar con comilla simple, si abro con comilla doble, debo cerrar con comilla doble.

Para ejemplificar lo mencionado anteriormente, creare la misma variable pero abriré con comilla doble y cerrare con comilla simple.

```
>>> ip = "192.168.1.1'
  File "<stdin>", line 1
    ip = "192.168.1.1'
                      ^
SyntaxError: EOL while scanning string literal
>>>
```

Podemos apreciar que el intérprete de Python, nos arroja un error de sintaxis, afirmando lo explicado anteriormente.

Ahora si queremos comprobar que tipo de datos contiene la variable ip utilizaremos la función **type()** de la siguiente manera:

```
>>> type(ip)
<type 'str'>
>>>
```

En el ejemplo anterior, el interprete con indica que es de tipo texto con la abreviación **str** que significa **string**.

Ahora vamos a probar como se escribiría en python la variable puerto:

```
>>> puerto = 443
>>> print puerto
443
>>> type(puerto)
<type 'int'>
>>>
```

Realizando el mismo ejemplo que en el ejercicio anterior, nos damos cuenta que esta vez, no se utilizo comillas, esto se debe a que los valores numéricos se escriben sin comillas y que al aplicarle la función **type()** nos entrega que es de tipo **int** que es la abreviación la palabra integer.

Para terminar con este ejemplo, utilizaremos una variable de tipo Verdadero o Falso:

```
>>> conexion = True
>>> print conexion
True
>>> type(conexion)
<type 'bool'>
>>>
```

El ejemplo nos indica que en la variable conexión tiene un valor de tipo bool que es la abreviación de boolean.

type(variable) Retorna el tipo de una variable y es muy útil para depurar programas cuando no sabemos con que tipo de datos estamos trabajando.

Manejo de String o cadenas de texto

Aprender a manipular de cadenas de texto es muy importante para cualquier programador, En este capitulo intentaremos revisar la mayor parte de ejemplos posibles, para lograr un dominio aceptable y después nos sea fácil realizar cualquier tipo de tarea.

Manipulación de subcadenas:

Para manipular subcadenas tenemos que imaginarnos que cada cadena de texto se puede descomponer en pequeños trozos dependiendo de lo que queramos hacer.

En el siguiente ejemplo vamos a demostrar que un conjunto de textos agrupados suman una cadena de caracteres:

```
>>> sitio = 'https://www.python.org'
>>> len(sitio)
22
```

Se utiliza la función nativa de python **len()** para contar cuantos caracteres esta compuesta la variable sitio, en este caso nos dio 22 caracteres que lo podemos comprobar en el siguiente ejemplo:

h	t	t	p	s	:	/	/	w	w	w	.	p	y	t	h	o	n	.	o	r	g
0	1	2	3	4	5	6	7	8	9	10	11	12	13	14	15	16	17	18	19	20	21

Las variables en python comienzan en cero por lo que la suma en total es 22. Si queremos saber el valor de cada posición debe ser de la siguiente manera:

```
>>> print sitio[0]
h
>>> print sitio[1]
t
>>> print sitio[2]
t
>>> print sitio[3]
p
>>> print sitio[4]
s
>>> print sitio[5]
:
>>> print sitio[6]
/
>>> print sitio[7]
/
>>>
```

Nombre de la variable que queremos manipular + símbolo [número] nos indica la posición del texto (**sitio [número])**. En el ejercicio anterior listamos desde la posición 0 a la posición numero 7. Con esto fuimos recorriendo letra por letra el texto, ahora si queremos extraer todo lo que esta en la posición 0 a 7 vasta con escribir [0:8] esto nos devolverá el siguiente texto:

h	t	t	p	s	:	/	/
0	1	2	3	4	5	6	7

El ejemplo aplicado en la consola interactiva se va ver de la siguiente manera:

```
>>> print sitio[0:8]
https://
>>>
```

De la variable sitio obtengamos el protocolo utilizado y el nombre de dominio:

```
>>> protocolo = sitio[0:8]
>>> dominio = sitio[8:22]
>>> print "El protocolo utilizado es :" + protocolo
El protocolo utilizado es :https://
>>> print "El nombre de dominio es : " + dominio
El nombre de dominio es : www.python.org
>>>
```

Concatenación de variables e interpolación

La concatenación en la programación se trata de unir o juntar dos variables. Para poder unir dos valores en python se puede utilizar el símbolo mas (+) el símbolo coma (,) o la interpolación.

Concatenación con el símbolo (+)

```
>>> protocolo = "https://"
>>> sitio = "www.python.org"
>>> print protocolo + sitio
https://www.python.org
>>>
```

Concatenación con el símbolo (,)

```
>>> protocolo = "https://"
>>> sitio = "www.python.org"
>>> print protocolo , sitio
https:// www.python.org
>>>
```

Interpolación

```
>>> protocolo = "https://"
>>> sitio = "www.python.org"
>>> print "El protocolo es {0} y el dominio {1} ".format(protocolo,sitio)
El protocolo es https:// y el dominio www.python.org
>>>
```

En el ejemplo de la interpolación nos damos cuenta que se utiliza la función **format()** que nos permite reemplazar los valores en una cadena de texto, el símbolo**{número}** indica que el valor se escribirá en esa parte de la cadena de texto.

Texto multilíneas

Para trabajar con texto multilíneas se debe utilizar triple comilla (“”””) como en el siguiente ejemplo:

```
>>> multilinea = """
... esto es la primare linea
... la segunda linea
...
...
... fin lineas """
```

Imprimimos la variable:

```
>>> print multilinea

esto es la primare linea
la segunda linea


fin lineas
>>>
```

Uso de comentarios

Para realizar comentarios en el código fuente sin que sea leído por el intérprete de Python debemos utilizar el carácter gato (#) de la siguiente manera:

```
>>> # este es un comentario
```

Los comentarios en el código fuente son útiles para saber de que se trata el código fuente, especificar para que se utilicen las variables o cualquier cosa que queramos escribir.

Saltos de líneas

El símbolo que debemos usar para crear un salto de línea es el carácter **\n**

Ejemplo:

```
>>> head = " HEAD / HTTP/1.1 \n Host: python.org \n connection: keep-alive"
>>> print head
 HEAD / HTTP/1.1
 Host: python.org
 connection: keep-alive
>>>
```

El resultado al imprimir la variable head es con dos saltos de línea.

Funciones útiles en la manipulación de texto.

Función **lower ()**: pasa una cadena de testo a letra minúscula.

Ejemplo:

```
>>> sitio ="HTTP://WWW.PYTHON.ORG"
>>> print sitio.lower()
http://www.python.org
>>>
```

Función **upper()**: Pasa una cadena de texto a Mayúscula.

Ejemplo:

```
>>> titulo = "este es un titulo"
>>> print titulo.upper()
ESTE ES UN TITULO
>>>
```

Función **str():** convierte una variable a tipo texto.

Ejemplo:

```
>>> puerto = 8080
>>> str(puerto)
'8080'
>>>
```

En el ejemplo podemos verificar que ingresamos una variable numérica y la convertimos en texto.

Función **replace()**: esta función permite buscar una cadena de texto y modificarla.

Ejemplo:

```
>>> buscar = "google.com"
>>> reemplazar = "python.org"
>>> texto = "http://www.google.com"
>>> print texto.replace(buscar,reemplazar)
http://www.python.org
>>>
```

En el ejemplo anterior se entiende que la función **replace,** el primer parámetro recibe lo que se desea buscar (puede ser palabra o letra) y el segundo parámetro es el texto que se desea reemplazar. Es decir busca el dominio **google.com** y reemplázalo por **python.org**.

Función **count()**: esta función permite saber cuantas veces se repite una letra o texto en una variable

Ejemplo:

```
>>> sitio = "www.python.org"
>>> print sitio.count('w')
3
>>>
```

Si en algún momento tenemos problemas al utilizar caracteres extraños en nuestro programa y nos sale un error parecido a este.

```
SyntaxError: Non-ASCII character
```

Debemos incluir en nuestro programa la siguiente instrucción en la cabecera o primera línea de codigo:

```
# -*- coding: utf-8 -*-
```

Números en Python

Tipo Enteros

Cuando hablamos de entero o integer en ingles, nos estamos refiriendo a valores numéricos positivos y negativos {…, -3, -2, -1, 0, 1, 2, 3,…}, no decimales.

```
>>> entero = 12
>>> type(entero)
<type 'int'>
>>>
```

Tipo Long

Los números tipo long tienen mas precisión que los números enteros, se recomienda utilizar este tipo de variables cuando se hagan cálculos complejos que se necesite exactitud. En los tipo long podemos almacenar números de -2^31 a 2^31 – 1. Se trabaja con tipo long poniendo una L al final.

```
>>> entero = 12L
>>> type(entero)
<type 'long'>
>>>
```

Tipo Octal

Trabajaremos con números en base 8 cuando al principio del valor ponemos un cero.

```
>>> entero = int(041)
>>> type(entero)
<type 'int'>
>>> print entero
33
>>>
```

En este ejercicio llama la atención que utilice una nueva función para transformar a entero un dato, en este caso 041 en base 8 es 33 en base 10.

Tipo Hexadecimal

Para trabajar con hexadecimales, tenemos que anteponer el símbolo **0x.**

```
>>> entero = 0x77
>>> print int(entero)
119
>>> print hex(119)
0x77
>>>
```

Nuevamente vemos una nueva función que podemos utilizar, su nombre es **hex()** y convierte una variable a hexadecimal.

Tipo Reales

Los números reales son los decimales y estos se definen con el símbolo **punto (.)**

```
>>> decimal = 0.0003333
>>> type(decimal)
<type 'float'>
>>>
```

Operadores Aritméticos

Operador	Descripción	Ejemplo
+	Suma	num = 3 + 4
-	Resta	num = 7 – 4
	Negación	num = -7
*	Multiplicación	num = 3 * 4
**	Exponente	num = 3 ** 4
/	División	num = 3.7 / 2
//	División entera	num = 3.7 // 2
%	Módulo	num = 7 % 3

Valores Boléanos

El siguiente ejemplo que vamos a imaginar que hacemos dos comparaciones con la conjunción Y.

```
>>> True and False
False
>>> False and False
False
>>> False and True
False
>>> True and True
True
>>>
```

Las conjunciones con la palabra reservada **and** significan que se debe realizar una multiplicación:

1 significa verdadero
0 significa falso.

True and False	**False**	**1 x 0**	**0**
False and False	**False**	**0 x 0**	**0**
False and True	**False**	**0 x 1**	**0**
True and True	**True**	**1 x 1**	**1**

Las conducciones con la palabra reservada **or** significan suma.

```
>>> True or False
True
>>> False or True
True
>>> True or True
True
>>> False or False
False
>>>
```

Tabla de comparaciones posibles soluciones:

True or False	**True**	**1 + 0**	**1**
False or False	**False**	**0 + 0**	**0**
False or True	**True**	**0 + 1**	**1**
True or True	**True**	**1 + 1**	**1**

Operadores Relacionales

Muchas veces vamos a tener que comparar dos o mas valores, python nos entrega la siguiente herramienta:

Operador	Descripción	Ejemplo	Resultado
==	¿Son iguales 7 y 3?	7 == 3	False
!=	¿Son distintos 7 y 3?	7 != 3	True
<	¿7 es menor que 3?	7 < 3	False
>	¿7 es mayor que 3?	7 > 3	True
<=	¿7 es menor o igual que 7?	7 <= 7	True
>=	¿7 es mayor o igual que 3?	7 >= 3	True

Ahora que tenemos claro los posibles resultados, vamos a realizar un ejemplo con lo aprendido hasta ahora:

```
>>> ip = "192.168.1.3"
>>> puerto = 443
>>> conexion = True
>>> if conexion:
...     print "Conectando a la ip " + ip + " puerto " + str(puerto)
...
Conectando a la ip 192.168.1.3 puerto 443
>>>
```

En las primeras líneas del código se declara la variable ip, puerto y conexión, luego se pregunta si conexión es Verdadera, entonces imprime en pantalla conectando a la ip 192.168.1.3 puerto 443. El ejemplo es súper sencillo, pero se puede entender para que nos podría servir lo aprendido hasta ahora, en el ámbito de la seguridad informática.

Agrupación de Datos

05

Agrupación de Datos

Muchas veces cuando estamos programando, necesitamos guardad muchos datos en una sola variable, para posteriormente listarlos o trabajar con ellos. La agrupación de datos es muy parecida a una bodega donde se guardan y se sacan cosas, la bodega en este caso seria la memoria RAM y las cosas que guardamos serian nuestros datos. Python provee 3 formas distintas de trabajar con agrupación de variables:

- Tuplas
- Listas
- Diccionarios

Estas 3 formas de agrupar datos se diferencian por su sintaxis y su manera de ser manipulados.

Tuplas

Las tuplas son listas **inmutables** lo que significa que No puede ser cambiado o alterado después de su creación. Para definir una variable como Tupla se debe escribir el nombre de la variable signo igual y entre paréntesis **()** los valores a guardar separado por el símbolo coma (**,**).

Ejemplo:

```
alumno@hackerpython:~$ python
Python 2.7.6 (default, Mar 22 2014, 22:59:38)
[GCC 4.8.2] on linux2
Type "help", "copyright", "credits" or "license" for more information.
>>>
>>> mi_tupla = ("192.168.1.1",80,"192.168.1.2",21)
>>>
```

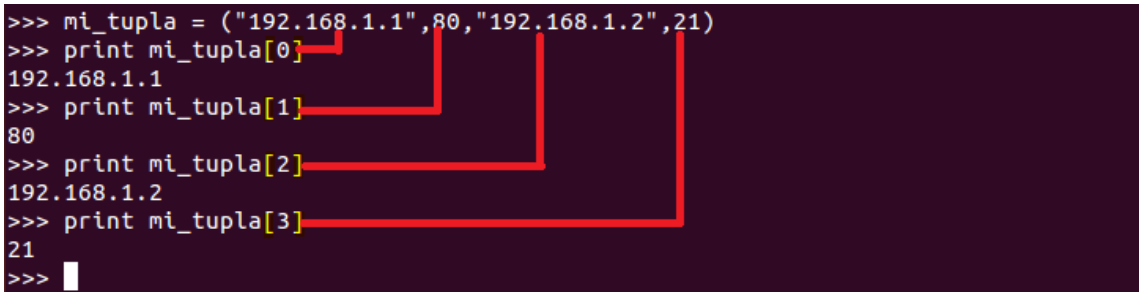
En la imagen se puede apreciar que una tupla soporta múltiples tipos de variables.

Para acceder a los valores almacenados se debe escribir el nombre de la variable acompañado de **[]** y en su interior la posición que se desea acceder.

Ejemplo:

```
>>> mi_tupla = ("192.168.1.1",80,"192.168.1.2",21)
>>> print mi_tupla[0]
192.168.1.1
>>> print mi_tupla[1]
80
>>> print mi_tupla[2]
192.168.1.2
>>> print mi_tupla[3]
21
>>>
```

Visualmente lo visto anteriormente se puede representar de la siguiente manera:



Otra forma en la que se podría acceder a los datos:

```
>>> print mi_tupla[0:4]
('192.168.1.1', 80, '192.168.1.2', 21)
>>> print mi_tupla[0:3]
('192.168.1.1', 80, '192.168.1.2')
>>> print mi_tupla[:2]
('192.168.1.1', 80)
>>> print mi_tupla[3:]
(21,)
>>>
```

En el siguiente ejemplo comprobamos que no se pueden asignar valores a las tuplas ya que son inmutables:

```
>>> mi_tupla[3] = "192.168.1.3"
Traceback (most recent call last):
  File "<stdin>", line 1, in <module>
TypeError: 'tuple' object does not support item assignment
>>>
```

Listas

Las listas a diferencia de las tuplas permiten su modificación una vez creados, es decir se pueden eliminar valores, modificar, crear y buscar. Su forma de uso es parecido a las tuplas solo que en vez de el signo paréntesis, se utiliza los corchetes **[].**

Ejemplo:

```
alumno@hackerpython:~$ python
Python 2.7.6 (default, Mar 22 2014, 22:59:38)
[GCC 4.8.2] on linux2
Type "help", "copyright", "credits" or "license" for more information.
>>> lista =["192.168.1.1",80,"192.168.1.2",80]
>>>
```

Para acceder a los datos seria de la siguiente manera:

```
>>> lista =["192.168.1.1",80,"192.168.1.2",80]
>>> print lista[0]
192.168.1.1
>>> print lista[1]
80
>>> print lista[2]
192.168.1.2
>>> print lista[3]
80
>>>
```

Otra forma de acceder a los datos es utilizando índices negativos:

```
>>> print lista[-1]
80
>>> print lista[-2]
192.168.1.2
>>> print lista[-4]
192.168.1.1
>>>
```

Como se ve en la imagen los índices negativos se accede a los datos de izquierda a derecha.

Agregar elementos

La forma de agregar elementos a una lista es utilizando el método **append()** , **insert()** y **extend().**

El Método append() agrega el elemento al final de la fila.

```
>>> lista.append("192.168.1.3")
>>> print lista
['192.168.1.1', 80, '192.168.1.2', 80, '192.168.1.3']
>>>
```

El Método insert() agrega un elemento a una posición en especifico en una fila.

```
>>> lista.insert(3,"192.168.1.44")
>>> print lista
['192.168.1.1', 80, '192.168.1.2', '192.168.1.44', 80, '192.168.1.3']
>>>
```

El método extend() concatena dos listas

```
>>> lista.extend([21,"192.168.1.77"])
>>> print lista
['192.168.1.1', 80, '192.168.1.2', '192.168.1.44', 80, '192.168.1.3', 21, '192.1
68.1.77']
>>>
```

Eliminar elementos

Para eliminar elementos en una lista python nos entrega el método **remove()** y su uso es de la siguiente forma:

```
>>> lista.remove("192.168.1.44")
>>> lista.remove("192.168.1.77")
>>> lista.remove("192.168.1.1")
>>> lista.remove("192.168.1.2")
>>> print lista
[80, 80, '192.168.1.3', 21]
>>>
```

Si necesitamos saber si en la lista existe un valor en especifico podemos utilizar el método **index().** Su uso es de la siguiente forma**:**

```
>>> print lista
[80, 80, '192.168.1.3', 21]
>>> print lista.index("192.168.1.3")
2
>>> print lista.index(21)
3
>>>
```

Como nos damos cuenta en la imagen anterior se nos indica la posición del índice donde se encuentra la variable. Pero que pasa si buscamos un valor que ya no se encuentre en la lista.

```
>>> print lista.index("192.168.1.44")
Traceback (most recent call last):
  File "<stdin>", line 1, in <module>
ValueError: '192.168.1.44' is not in list
>>>
```

Si un valor no se encuentra en la lista el intérprete de python nos arrojara un error, esto quiere decir que la función index() no nos sirve para buscar datos ya que se cae el programa, para esto Python tiene una función útil que se llama **in y** se utiliza de la siguiente manera:

```
>>> print lista.index("192.168.1.44")
Traceback (most recent call last):
  File "<stdin>", line 1, in <module>
ValueError: '192.168.1.44' is not in list
>>> "192.168.1.44" in lista
False
>>>
```

```
>>> "192.168.1.3" in lista
True
>>>
```

El ejemplo anterior se lee de la siguiente manera:

- En la variable lista existe un valor de tipo texto que contiene “192.1.68.1.44”, la función **in** retornara un valor False.

- En la variable lista existe un valor de tipo texto que contiene “192.1.68.1.3”, la función **in** retornara un valor True.

Cuando no sepamos si un elemento esta o no esta en una lista utilizaremos la función **in.**

Las listas también pueden ser concatenadas con el símbolo +.

Ejemplo:

```
>>> puertos = [21,22,80,443]
>>> puertos + [8080,3306]
[21, 22, 80, 443, 8080, 3306]
>>>
```

Diccionarios

Un diccionario es una forma muy parecida a implementar listas, pero con la diferencia que en vez de acceder a un dato por su incide, lo hacemos por medio de una palabra clave, es decir cada palabra clave debe ir asociada a un valor. Otra cosa que se diferencia de una lista es que se define un diccionario con el carácter **{}.**

Ejemplo:

```
>>> dicc = {"victima":"192.168.1.33", "atacante":"192.168.1.44"}
>>>
>>> dicc = {"victima":"192.168.1.33", "atacante":"192.168.1.44"}
>>> print dicc["victima"]
192.168.1.33
>>> print dicc["atacante"]
192.168.1.44
>>>
```

Como se puede apreciar en la imagen anterior, cada par de **clave: valor** van con dos puntos **(:)** entre medio y se separan por el símbolo coma **(,)**. Para ver los valores de una diccionario se debe escribir el nombre de la clave entre corchetes.

Para eliminar una entrada se debe utilizar la función **del ()** de la siguiente manera:

```
>>> del(dicc["atacante"])
>>> print dicc
{'victima': '192.168.1.33'}
>>>
```

Para volver a asignar un dato dentro de un diccionario se realiza de la siguiente forma:

```
>>> print dicc
{'victima': '192.168.1.33'}
>>> dicc['victima'] = "192.168.1.77"
>>> print dicc['victima']
192.168.1.77
>>>
```

Ahora utilicemos un diccionario pensando en un objeto con distintos atributos. El objeto va ser un sitio Web y sus atributos van a ser las tecnologías utilizadas:

Ejemplo:

```
>>> sitio = {}
>>> sitio = {"url":"www.algo.org","servidor" : "Apache 2.2.", "lenguaje" : "php5
", "libreria" : "Query, Angular","DB" : "PostgreSQL"}
>>> print sitio['url']
www.algo.org
>>> print sitio['servidor']
Apache 2.2.
>>> print sitio['DB']
PostgreSQL
>>>
```

Con este ejemplo nos damos cuenta que utilizar diccionarios tiene sentido cuando tratamos de programar lo mas parecido a como nosotros los humanos procesamos las cosas.

Si queremos saber el numero de elementos que tiene un diccionario utilizamos la función **len ()**

```
>>> len(sitio)
5
>>>
```

Si queremos saber que palabras claves tiene un diccionario utilizamos la función **keys()**

```
>>> sitio.keys()
['url', 'servidor', 'DB', 'lenguaje', 'libreria']
>>>
```

Si queremos conocer solamente los valores de un diccionario utilizamos la función **values ()**

```
>>> sitio.values()
['www.algo.org', 'Apache 2.2.', 'PostgreSQL', 'php5', 'Query, Angular']
>>>
```

Otro forma de obtener un valor es utilizando la función **get()** con su clave

```
>>> sitio.get('url')
'www.algo.org'
>>>
```

Para eliminar un valor según su clave se utiliza la función **pop ()** y valor clave

```
>>> sitio
{'url': 'www.algo.org', 'servidor': 'Apache 2.2.', 'DB': 'PostgreSQL', 'lenguaje
': 'php5', 'libreria': 'Query, Angular'}
>>> sitio.pop('libreria')
'Query, Angular'
>>> sitio
{'url': 'www.algo.org', 'servidor': 'Apache 2.2.', 'DB': 'PostgreSQL', 'lenguaje
': 'php5'}
>>>
```

Para copiar un diccionario utilizaremos la función **copy()**

```
>>> copia_sitio = sitio.copy()
>>> copia_sitio
{'url': 'www.algo.org', 'servidor': 'Apache 2.2.', 'DB': 'PostgreSQL', 'lenguaje
': 'php5'}
>>>
```

Para trasformar a tuplas las claves: valores del diccionario podemos utilizar la función **ítems()**

```
>>> sitio
{'url': 'www.algo.org', 'servidor': 'Apache 2.2.', 'DB': 'PostgreSQL', 'lenguaje
': 'php5'}
>>> sitio.items()
[('url', 'www.algo.org'), ('servidor', 'Apache 2.2.'), ('DB', 'PostgreSQL'), ('l
enguaje', 'php5')]
>>>
```

Para eliminar los elementos de un diccionario utilizamos la función **clear()**

```
>>> sitio.clear()
>>> sitio
{}
>>>
```

Asignación Múltiple

Python dentro de sus múltiples funciones permite asignar varios valores en una línea de código, es decir, se puede asignar por ejemplo de una lista, tupla o valores separados por coma(,).

Ejemplo múltiples variables:

```
>>> host,directorio = "http://www.sitio.cl","/admin/"
>>> print host + directorio
http://www.sitio.cl/admin/
>>>
```

Ejemplo Listas:

```
>>> lista = ["192.168.1.33","HTTP",80]
>>> ip,protocolo,puerto = lista
>>> print ip
192.168.1.33
>>> print protocolo
HTTP
>>> print puerto
80
>>>
```

En el código fuente se puede apreciar que las variables de la Lista pasan directamente a las tres variables según su orden ip, protocolo, puerto. Esto es lo mismo que escribir:

Ip = “192.168.1.33”

Protocolo = “HTTP”

Puerto = 80

Ejemplo Listas:

```
>>> sitio = ("http://www.sitio.cl","/admin/")
>>> host,directorio = sitio
>>> print host
http://www.sitio.cl
>>> print directorio
/admin/
>>> print host + directorio
http://www.sitio.cl/admin/
>>>
```

Unión y división de cadenas de texto

Python nos entrega funciones tanto para unir cadenas de texto como para separar. En los siguientes ejemplos veremos como se realizan estas tareas.

Método join()

Para unir cadenas de texto podemos utilizar el método **join()** que nos ayuda a dar formato a un texto en particular.

Ejemplo Tupla:

```
>>> tupla = ("www.sitio.cl","producto","id","33")
>>> unir = "/"
>>> unir.join(tupla)
'www.sitio.cl/producto/id/33'
>>>
```

Como se puede apreciar en el código fuente, se une la tupla en una sola cadena de texto iterando el carácter **/ .**

Ejemplo Tupla 2:

```
>>> "?".join(("www.sitio.cl/producto/id.php","33"))
'www.sitio.cl/producto/id.php?33'
>>>
```

Este código es un poco parecido al ejemplo anterior, solo que en vez de definir una tupla, se ingresa directamente como parámetro los valores a la función join().

Ejemplo cadena de texto:

```
>>> "-".join("1234567")
'1-2-3-4-5-6-7'
>>>
```

En este ejemplo se puede ver que si se ingresa como parámetro una cadena de texto a la función **join(),** esta separara cada carácter por el texto que nosotros le indiquemos.

Ejemplo utilizando listas:

```
>>> db = ['server=sitio.cl', 'user=root', 'database=master', 'password=']
>>> ";".join(db)
'server=sitio.cl;user=root;database=master;password='
>>>
```

Método split()

El método **split()** sirve para separar una cadena por medio de algún parámetro que le pasemos.

```
>>> texto = "www.sitio.cl/admin/user/tesla"
>>> texto.split("/")
['www.sitio.cl', 'admin', 'user', 'tesla']
>>> resultado = texto.split("/")
>>> type(resultado)
<type 'list'>
>>>
```

En el código anterior se comprueba que el método **split()** retorna una lista con los valores separados.

Otra forma en que se puede utilizar este método es en una sola línea de código:

```
>>> "www.sitio.cl/admin/user/tesla".split("/")
['www.sitio.cl', 'admin', 'user', 'tesla']
>>>
```

Método splitlines()

El método splitlines() nos sirve para separar una cadena de texto que contiene varias líneas.

```
>>> diccionario =""" admin
... password
... root
... postgres
... user
... admin123
... admin1234"""
>>> diccionario.splitlines()
[' admin', 'password', 'root', 'postgres', 'user', 'admin123', 'admin1234']
>>>
```

En el código fuente se comprueba que el retorno de la función es una lista.

Estructuras de Control IF, ELSE y ELIF

06

Estructuras de Control IF, ELSE y ELIF

Las estructura de control condiciona, son la forma en que un lenguaje de programación nos entrega la posibilidad de hacer preguntas y comparaciones a la maquina, esto nos ayuda enormemente cuando queremos dar inteligencia a nuestros programas.

If: La instrucción if(si condicional), sirve para verificar o comprobar que un suceso si ocurrió y lleva al final de la operación el símbolo(**:**).

else: La instrucción else(de lo contrario) puedo o no acompañar al if y sirve para comprobar que no se cumplió la comparación o verificación, lleva al final el símbolo(**:**).

Uso:

```
if expresion:
  tarea_a_realizar()
else:
  tarea_a_realizar()
```

Ejemplo:

```
>>> res = [21,'FTP','ProFTPD 1.3.1',True]
>>> if res[3]:
...     if res[2] == "ProFTPD 1.3.1":
...         print "El servicio " + res[1] + " es vulnerable"
...         print ">> exploit/unix/ftp/proftpd_133c_backdoor"
...     else:
...         print "No se encontro exploit"
... else:
...     print "El puerto esta cerrado"
...
El servicio FTP es vulnerable
>> exploit/unix/ftp/proftpd_133c_backdoor
>>>
```

Lo primero que realiza el programa, es verificar el resultado de una lista y comprobar el índice en la posición 3, este contiene un valor que hace referencia a si hubo conexión o no con el servidor (Verdadero /Falso).

Si la respuesta es True (Verdadera), vuelve a realizar una segunda comparación, para esto se toma de la lista el índice 2 y se comprueba si el banner entregado por el servicio es igual (==) a ProFTPD 1.3.1. Luego de esta operación, se imprime en pantalla el servicio auditado y el exploit que se puede lanzar para tomar control de la maquina.

Caso contrario de la segunda comparación (else), indica que no hubo coincidencia por lo tanto desconocemos si existe un exploit.

Caso contrario de la primera comparación (else), indica que en la lista tenemos un False por lo que no hubo conexión con el servidor.

elif: La instrucción elif (de lo contrario si) es parecida al if, la diferencia que sirve para una seguir haciendo comparaciones y lleva al final de la operación el símbolo(**:**).

```
if comparacion1:
   print “se cumplió la condición 1”
elif comparacion2:
     print “se cumplió la condición 2”
elif comparacion3:
     print “se cumplió la condicion3”
else:
     print “no tuvimos éxito en la comparación”
```

Ejemplo:

```
>>> puerto = 3389
>>> if puerto == 21:
...     print "Servicio FTP"
... elif puerto == 22:
...     print "Servicio SSH"
... elif puerto == 80:
...     print "Servicio HTTP"
... elif puerto == 443:
...     print "Servicio HTTPS"
... elif puerto == 3306:
...     print "Servicio MySQL"
... elif puerto == 3389:
...     print "Servicio RDP"
... elif puerto == 5432:
...     print "Servicio PostgreSQL"
... else:
...     print "Servicio no identificado"
...
Servicio RDP
```

El programa visto anteriormente como ejemplo, demuestra una comparación múltiple para verificar que servicio pudiese estar corriendo en un servidor solo indicando el número del puerto.

Como habrán notado en los ejemplos vistos hasta ahora, los códigos fuentes son bastante ordenados. Una cosa muy necesaria en Python para usar estructura de control, es respetar la identación o estructura gramatical, es decir, el intérprete de Python necesita que después de un if, te saltes cuatro espacios y escribas lo que necesitas que ocurra en el programa.

Indentación es un anglicismo (de la palabra inglesa indentation) de uso común en informática; no es un término reconocido por la Real Academia Española . La Real Academia recomienda utilizar «sangrado». Este término significa mover un bloque de texto hacia la derecha insertando espacios o tabuladores, para así separarlo del margen izquierdo y mejor distinguirlo del texto adyacente; en el ámbito de la imprenta, este concepto siempre se ha denominado sangrado o sangría.

Para poder realizar otros tipos de de tareas y complementar la instrucción if tenemos los siguientes Operadores:

Los operadores binarios:

&	AND
|	OR
^	XOR
~	NOT

Los Operadores de comparación o relacionales:

<	Menor que
>	Mayor que
<=	Menor o igual que
>=	Mayor o igual que
==	Igual a
!=	Distinto de

Cabe destacar que Python no tiene incluido dentro de su Core la función **switch**, en su caso podemos reemplazar con la estructura de control elif.

En esta página encontraras ejemplos prácticos de análisis forense, espero ir llenando de poquito con artículos interesantes y compartir conocimientos.

Bucle While y For

07

Bucle while

La palabra reservada **while** es un bucle que permite ejecutar reiteradas veces alguna acción, hasta que se cumpla la condición.

Es importante saber que cuando definimos un bucle while en nuestro código fuente, las iteraciones se van a cumplir siempre que la condición sea True o verdadera.

Ejemplo:

```
>>> incrementador = 1
>>> rango = 10
>>> while incrementador <= rango:
...     print ">> Atacando la IP " + "192.168.1." + str(incrementador)
...     incrementador = incrementador + 1
...
>> Atacando la IP 192.168.1.1
>> Atacando la IP 192.168.1.2
>> Atacando la IP 192.168.1.3
>> Atacando la IP 192.168.1.4
>> Atacando la IP 192.168.1.5
>> Atacando la IP 192.168.1.6
>> Atacando la IP 192.168.1.7
>> Atacando la IP 192.168.1.8
>> Atacando la IP 192.168.1.9
>> Atacando la IP 192.168.1.10
>>>
```

El ejemplo anterior tiene dos variables definidas:

- La primera variable es un incrementador y esta definida como un numero 1.
- La variable rango en el programa se define como numérica con el valor 10 y tiene que ver con el limite de iteraciones.

La tercera línea se define la condición y se lee de la siguiente manera, mientras incrementador (valor 1) sea menor (<) o igual (=) a rango (valor 10), repite lo que esta dentro de la identación hasta que se cumpla la condición y terminado por dos puntos (**:**)

Procedemos a crear una identación de 4 espacios para marcar el principio y el fin del ciclo while.

La cuarta línea se imprime en pantalla con cada iteración el mensaje variando solo la dirección IP.

La cuarta línea es muy importante ya que se utiliza la variable incrementador, que va sumando 1 con cada ciclo. Es decir, la variable incrementador vale 2 por que se sumo incrementador que vale 1 + 1. el ciclo se repite hasta que 10 <= 10.

Acumulador: Una forma muy sencilla de entender un acumulador es hacer el siguiente ejercicio en la Terminal.

```
>>> incrementador = 1
>>> print incrementador
1
>>> incrementador = incrementador + 1
>>> print incrementador
2
>>> incrementador = incrementador + 1
>>> print incrementador
3
>>>
```

Otra forma en la que podemos encontrar un acumulador es la siguiente:

```
>>> incrementador += 1
>>> print incrementador
4
>>>
```

Bucle for

En Python el bucle for, principalmente nos permite iterar sobre variables de o lista o tupla, en el siguiente ejemplo vamos a realizar un ejercicio para que nos quede mas claro como lo podemos utilizar.

Bucle en Tupla

```
>>> hosts = ('www.google.cl','www.python.org','www.w3c.org')
>>> for host in hosts:
...     print "buscando en : " + host
...
buscando en : www.google.cl
buscando en : www.python.org
buscando en : www.w3c.org
>>>
```

En la primera línea de código definimos una tupla con varios sitios Web.

La segunda línea declaramos el ciclo for, lo sigue la variable host donde guardamos lo que sacamos de la tupla, terminado por dos puntos (**:**).

El ciclo for va recorriendo una a una las posiciones de la tupla hasta que ya no encuentre nada, lo que en programación se interpretaría como un False

Después de declarar el ciclo se crea un espacio de identación que en este caso es de 4 espacios.

Luego se imprime en pantalla cada vez que se recorra la Tupla.

Bucle en una Lista

Trabajar con Listas es muy parecido a las tuplas por lo que podemos aplicar el mismo ejemplo que vimos anteriormente.

```
>>>
>>> hosts = ['www.google.cl','www.python.org','www.w3c.org']
>>> for host in hosts:
...     print "buscar en : " + host
...
buscar en : www.google.cl
buscar en : www.python.org
buscar en : www.w3c.org
>>>
```

Como se puede apreciar en el código, la única diferencia es el uso de corchetes que diferencia una lista de una tupla.

Otra forma de realizar bucles for es utilizando el método **range ()** que provee Python.

Ejemplo método range:

```
>>> range(5)
[0, 1, 2, 3, 4]
>>> range(3,7)
[3, 4, 5, 6]
>>> range(1,10,2)
[1, 3, 5, 7, 9]
>>> rango = range(1,10)
>>> type(rango)
<type 'list'>
>>>
```

En el código anterior el método **range([start], stop[, step])** puede recibir 3 parámetros:

- **start** número de la secuencia.
- **stop:** Generar números hasta
- **step:** Diferencia entre cada número en la secuencia.

Además el método **range()** retornas una **lista** y lo podemos comprobar con la función **type().**

Ejemplo:

```
>>> for host in range(1,3):
...     print ">> Atacando la IP : " + "192.168.1." + str(host)
...
>> Atacando la IP : 192.168.1.1
>> Atacando la IP : 192.168.1.2
>>>
```

Funciones en Python

08

Funciones en Python

Las funciones en cualquier lenguaje de programación, se inventaron para ayudarnos a simplificar tareas y para que podamos reutilizar código fuente, antiguamente se programaba de forma estructurada repitiendo una y otra vez las instrucciones a medida que un programa avanzaba. Ahora podemos agrupar un conjunto de instrucciones en una función y reutilizarla cuando nosotros queramos. En palabras simples una función es un pequeño trozo de programa que realiza una tarea en específico.

Las funciones en Python se declaran utilizando la palabra reservada **def,** que le indica al interprete de python que se esta declarando una función, luego va el nombre de la función, que para nuestra conveniencia, tiene que dar sentido a la tarea que realiza, luego para finalizar se utilizan dos puntos (**:**) y finalmente, se utiliza la identación de 4 espacios para marcar lo que se ejecuta dentro de esa función.

Ejemplo:

```
>>> def titulo_programa():
...     print "=========================="
...     print "  Titulo del Programa     "
...     print "=========================="
...
>>>
```

Como se aprecia en la imagen, no sucedió nada una vez que cargamos este pequeño trozo de programa. Cabe señalar que si queremos que se ejecute nuestra función la tenemos que llamarla por su nombre.

Ejemplo:

```
>>> def titulo_programa():
...     print "=========================="
...     print "  Titulo del Programa     "
...     print "=========================="
...
>>> titulo_programa()
==========================
  Titulo del Programa
==========================
>>>
>>>
```

Para llamar a una función, se debe escribir el nombre sin la palabra **def** y sin los dos puntos (**:**). La función la podemos llamar cuantas veces queramos y siempre va a realizar la misma tarea.

Ejemplo:

```
>>> titulo_programa()
=========================
  Titulo del Programa
=========================
>>> titulo_programa()
=========================
  Titulo del Programa
=========================
>>> titulo_programa()
=========================
  Titulo del Programa
=========================
>>>
```

Ahora como comprenderán, tener funciones en nuestros programas es muy útil, sobre todo cuando queremos realizar una tarea una y otra vez, sin tener que estar escribiendo código a cada rato. Estas funciones se les llama funciones void o vacías, ya que no retorna ninguna variable lo que podría limitar nuestros programas.

Funciones con retorno.

Las funciones con retornos nos pueden ser útiles, cuando necesitamos devolver algo al programa principal para su posterior tratamiento. Para hacerlas funcionar vamos a utilizar la palabra reservada **return** para devolver al programa algún resultado.

```
>>> def puertos():
...     puertos = [21,22,80,3389]
...     return puertos
...
>>> servicios = puertos()
>>> for servicio in servicios:
...     print "Atacando el puerto : " + str(servicio)
...
Atacando el puerto : 21
Atacando el puerto : 22
Atacando el puerto : 80
Atacando el puerto : 3389
>>>
```

El código fuente anterior nos muestra:

La función declarada guarda en un conjunto de puertos a atacar en la variable puertos, luego se retorna un objeto tipo list desde la función. Esto quiere decir que cada vez que llamemos a la función puertos, esta nos va a devolver una lista de puertos. Luego se recorre la lista con un ciclo for y se escribe por pantalla el puerto que se esta atacando.

Funciones con Parámetros

Hasta ahora, puede que todavía no encuentren útil el uso de funciones, yo les respondo que ahora se pone interesante la cosa, ya que las funciones con parámetros, nos permiten ejecutar tareas pero de forma dinámica, los parámetros son valores que necesita una función para poder funcionar, yo me lo imagino la función como una receta que siempre va a ser igual y los parámetros como distintos ingredientes.
Estas funciones se declaran con la misma estructura de siempre, solo que entre los paréntesis tenemos que indicar los valores que se necesitan.

Ejemplo:

```
>>> def atacar(ip,puertos):
...     for puerto in puertos:
...         print ">> Atacando a el Host " + ip + " Puerto " + str(puerto)
...
>>> atacar("192.168.1.33",[21,22,80,443,3306,3389])
>> Atacando a el Host 192.168.1.33 Puerto 21
>> Atacando a el Host 192.168.1.33 Puerto 22
>> Atacando a el Host 192.168.1.33 Puerto 80
>> Atacando a el Host 192.168.1.33 Puerto 443
>> Atacando a el Host 192.168.1.33 Puerto 3306
>> Atacando a el Host 192.168.1.33 Puerto 3389
>>>
```

En el ejemplo se define una función que supuestamente ataca a un servidor iterando por varios servicios, la función necesita dos argumentos o variables para realizar su tarea, pero también podría agregarse más argumentos si se requiere.
Ahora vamos a realizar otro ejemplo que supuestamente va a atacar, dependiendo del servicio que este corriendo en el servidor victima. Utilizaremos dos funciones y las llamaremos cuando las necesitemos:

Ejemplo:

```
>>> def ataqueFTP(ip):
...     print "Atacando al Host " + ip + "por FTP"
...
>>> def ataqueSSH(ip):
...     print "Atacando al Host " + ip + "por SSH"
...
>>> victimas = ["192.168.1.33","192.168.1.44"]
>>> puertos = [21,22,80,443]
>>> for victima in victimas:
...     for puerto in puertos:
...         if puerto == 21:
...             ataqueFTP(victima)
...         elif puerto == 22:
...             ataqueSSH(victima)
...         else:
...             print "No tenemos ataque para el puerto " + str(puerto)
...
Atacando al Host 192.168.1.33por FTP
Atacando al Host 192.168.1.33por SSH
No tenemos ataque para el puerto 80
No tenemos ataque para el puerto 443
Atacando al Host 192.168.1.44por FTP
Atacando al Host 192.168.1.44por SSH
No tenemos ataque para el puerto 80
No tenemos ataque para el puerto 443
>>>
```

Funciones con Parámetros por omisión

Muchas veces cuando estamos programando, se nos puede olvidar ingresar un parámetro en una función o podemos compartir nuestro código con otra persona y queremos dejar parámetros por defecto, para esto y más casos, nos serviría utilizar parámetros por omisión. Los parámetros por omisión se asignan valores inmediatamente si es que una función no recibe parámetros, pero no se le ingresa nada.

Ejemplo:

```
>>> def banner_aplicacion(Saludos = "HOLA AMIGOS"):
...     print "=========================="
...     print ""
...     print Saludos
...     print ""
...     print "=========================="
...
>>> banner_aplicacion()
==========================

HOLA AMIGOS

==========================
>>> banner_aplicacion("Software que escanea Puertos")
==========================

Software que escanea Puertos

==========================
>>>
```

Como podemos apreciar en la imagen, en el primer ejemplo al ejecutar la función sin ningún parámetro, se imprimió el valor por defecto “HOLA AMIGOS”. La segunda vez que ejecutamos la función y le asignamos un valor al parámetro, ejecuto lo que le indicamos en el.

Clases y Métodos en Python

09

Clases en Python

La programación orientada a objetos (POO, u OOP según sus siglas en inglés) es un paradigma de programación que viene a innovar la forma de obtener resultados. Los objetos manipulan los datos de entrada para la obtención de datos de salida específicos, donde cada objeto ofrece una funcionalidad especial.
Muchos de los objetos pre-diseñados de los lenguajes de programación actuales permiten la agrupación en bibliotecas o librerías, sin embargo, muchos de estos lenguajes permiten al usuario la creación de sus propias bibliotecas.

La programación orientada a objetos difiere de la programación estructurada tradicional, en la que los datos y los procedimientos están separados y sin relación, ya que lo único que se busca es el procesamiento de unos datos de entrada para obtener otros de salida. La programación estructurada anima al programador a pensar sobre todo en términos de procedimientos o funciones, y en segundo lugar en las estructuras de datos que esos procedimientos manejan. En la programación estructurada solo se escriben funciones que procesan datos. Los programadores que emplean POO, en cambio, primero definen objetos para luego enviarles mensajes solicitándoles que realicen sus métodos por sí mismos.

La POO es una forma de programar que trata de encontrar una solución a problemas de cualquier tipo. Introduce nuevos conceptos, que superan y amplían conceptos antiguos ya conocidos. Entre ellos destacan los siguientes:

Objeto
Instancia de una clase. Entidad provista de un conjunto de propiedades o atributos (datos) y de comportamiento o funcionalidad (métodos), los mismos que consecuentemente reaccionan a eventos. Se corresponden con los objetos reales del mundo que nos rodea, o con objetos internos del sistema (del programa).

Clase
Definiciones de las propiedades y comportamiento de un tipo de objeto concreto. La instanciación es la lectura de estas definiciones y la creación de un objeto a partir de ella.

Método
Algoritmo asociado a un objeto (o a una clase de objetos), cuya ejecución se desencadena tras la recepción de un "mensaje". Desde el punto de vista del comportamiento, es lo que el objeto puede hacer. Un método puede producir un cambio en las propiedades del objeto, o la generación de un "evento" con un nuevo mensaje para otro objeto del sistema.

Evento
Es un suceso en el sistema (tal como una interacción del usuario con la máquina, o un mensaje enviado por un objeto). El sistema maneja el evento enviando el mensaje adecuado al objeto pertinente. También se puede definir como evento la reacción que puede desencadenar un objeto; es decir, la acción que genera.

Atributos
Características que tiene la clase.

En términos simple una clase es una cosa, un método es lo que hace la cosa, los atributos son variables y las instancias son copias de un objeto con sus métodos. Ahora se podrían estar preguntando, que diferencia tiene un método a una función y la respuesta es que un método esta dentro de una clase y una función esta fuera. Me gustaría profundizar mucho más en este tema, pero me desenfocaría del objetivo principal por el cual escribo, que es enseñar a hacer herramientas de hacking.

Después de esta breve introducción vamos a ver unos ejemplos prácticos.

Las clases en python de definen con la palabra reservada **class** seguida del nombre genérico del objeto y finalizando con dos puntos (**:**)

Ejemplo clase:

```
class Ataque:
    …
    …
    …
```

Las propiedades del objeto Ataque podrían ser por ejemplo la dirección ip y el puerto a atacar.

Ejemplo Propiedades:

```
class Ataque:
    ip = '192.168.1.1'
    puerto = 80
    …
```

Los métodos son funciones o cosas que realiza el objeto.

Ejemplo métodos:

```
class Ataque:
    def __init__(self, ip, puerto):
      self.ip = ip
      self.puerto = puerto

    def FuerzaBrutaSSH(self):
        print 'SSH >> atacando al host ' + self.ip

    def FuerzaBrutaFTP(self):
        print 'FTP >> atacando al host ' + self.ip
```

Nótese el cambio que realice, el primer método **__init __ ()** es un método especial, se llama constructor de la clase o el método de inicialización. Es lo primero que se ejecuta cuando se instancia la clase.
La palabra reservada **self** hace referencia a el mismo objeto, es decir, se llama a si mismo, esto significa que cuando llamamos a **self.ip,** recogemos lo que esta dentro de la variable ip en el programa de ejemplo.

Ahora vamos a ver un ejemplo de instancia de la clase ataque:

Lo primero q vamos a hacer es utilizar un editor de texto para escribir nuestro código: gedit fbruta.py

fbruta.py

```
1  #!/usr/bin/python
2
3  class Ataque:
4      def __init__(self, ip, puerto):
5          self.ip = ip
6          self.puerto = puerto
7
8      def FuerzaBrutaSSH(self):
9          print 'SSH >> atacando al host ' + self.ip + ' puerto ' + str(self.puerto)
10
11     def FuerzaBrutaFTP(self):
12         print 'FTP >> atacando al host ' + self.ip + ' puerto ' + str(self.puerto)
13
14 atq = Ataque('192.168.1.1',22)
15 atq.FuerzaBrutaSSH()
16 atq.puerto = 21
17 atq.FuerzaBrutaFTP()
18 atq.ip = '192.168.1.3'
19 atq.puerto = 22
20 atq.FuerzaBrutaSSH()
```

De la línea 14 a la línea 20 se puede apreciar como mediante una clase, podemos repetir una tarea varias veces y modificar los elementos de un objeto.

Para ejecutar este código debemos escribir en nuestra Terminal la siguiente instrucción:

python ruta_archivo.py

```
alumno@hackerpython:~$ python /home/alumno/Escritorio/fbruta.py
SSH >> atacando al host 192.168.1.1 puerto 22
FTP >> atacando al host 192.168.1.1 puerto 21
SSH >> atacando al host 192.168.1.3 puerto 22
alumno@hackerpython:~$
```

En la imagen anterior se muestra el resultado de nuestro script.

Utilizando una clase como Librería en Python

10

Utilizando una clase como librería

Para el siguiente ejemplo vamos a utilizar el código creado anteriormente

fbruta.py

```
#!/usr/bin/python

class Ataque:
    def __init__(self, ip, puerto):
        self.ip = ip
        self.puerto = puerto

    def FuerzaBrutaSSH(self):
        print 'SSH >> atacando al host ' + self.ip + ' puerto ' + str(self.puerto)

    def FuerzaBrutaFTP(self):
        print 'FTP >> atacando al host ' + self.ip + ' puerto ' + str(self.puerto)

```

Luego creamos un nuevo archivo llamado importador.py y quedaría de la siguiente manera:

Ahora modificamos el archivo importador.py y escribiremos el siguiente código:

importador.py

```
from fbruta import Ataque

a = Ataque('192.168.1.1',21)
a.FuerzaBrutaFTP()
```

Como se aprecia en el código fuente anterior utilizamos dos nuevas palabras reservadas,
from : utilizamos from para indicarle al interprete de python que archivo vamos a utilizar, en este caso es el archivo fbruta.py pero lo escribimos sin su extensión.
import : utilizamos import para indicarle al interprete de python que objeto vamos a utilizar, en este caso la clase Ataque con sus distintos métodos.
En la linea 3 del código se hace una instancia de la clase y se heredan todos los métodos de la clase Ataque en la variable a.
En la linea 4 hacemos uso del método FuerzaBrutaFTP.
Para ejecutar este código basta con escribir el siguiente comando en la consola:

```
alumno@hackerpython:~$ python /home/alumno/Escritorio/clases/importador.py
FTP >> atacando al host 192.168.1.1 puerto 21
```

Entrada de Datos por Teclado

11

Entrada de datos por teclado

Python proporciona dos funciones integradas para leer una línea de texto de la entrada estándar por teclado, Estas funciones son:

- raw_input()
- input()

La función raw_input

El raw_input ([indicacion]) función lee una línea de la entrada estándar y lo devuelve como una cadena (quitando el salto de línea final).

Sintaxis :

```
str= raw_input(‘ingrese un host a atacar : ’)
```

```
>>> str = raw_input("Ingrese un host a atacar: ")
Ingrese un host a atacar: 192.168.1.3
>>> print "atacando a " + str
atacando a 192.168.1.3
>>>
>>>
```

Cabe señalar que la funcion raw_input() recibe y procesa de forma literal lo ingresado por teclado, por ejemplo si ingresamos 1+2 +4 el interprete lo maneja como texto y no lo procesa:

Ejemplo:

```
>>> str = raw_input("Ingrese algo por teclado: ")
Ingrese algo por teclado: 1+2+4
>>> print str
1+2+4
>>>
```

La Función input

La función input([indicacion]) es equivalente a raw_input, excepto que devuelve el resultado evaluado en su caso.

Ejemplo:

```
>>> str = input("Ingrese algo por teclado: ")
Ingrese algo por teclado: 1+2+4
>>> print str
7
>>>
```

Al recibir números, el interprete los proceso ejecutándolo como una operación matemática.

Otro ejemplo tratando de mostrar la diferencia que existe entre la función input y raw_input es ingresando por ejemplo una dirección ip:

```
>>> str = input("Ingrese algo por teclado: ")
Ingrese algo por teclado: 192.168.1.1
Traceback (most recent call last):
  File "<stdin>", line 1, in <module>
  File "<string>", line 1
    192.168.1.1
            ^
SyntaxError: invalid syntax
>>>
```

El intérprete al intentar procesar una dirección ip, se da cuanta que son números y los trabaja como tal, por lo que la función input, no sirve mucho para hacer nuestros programas, recomiendo utilizar casi siempre raw_input.

Manejo de Archivos en Python

12

Manejo de archivos en python

Python proporciona funciones y métodos necesarios para manipular archivos de forma predeterminada.

Función open()

Antes de leer o escribir en un archivo, hay que abrirlo utilizando la función open() . Esta función crea un objeto file que se utiliza para llamar a otros métodos de apoyo asociados con ella.

Sintaxis:

```
archivo = open(Nombre_archivo [, Modo_apertura][, Bufer])
```

Nombre_archivo: El argumento Nombre_archivo es un valor de cadena que contiene el nombre del archivo que se desea acceder.

Modo_apertura: Determina el modo en el que el archivo tiene que ser abierto, es decir, leer, escribir, añadir, etc. Este es un parámetro opcional y se lee el modo de acceso de archivos por defecto (r).

Búfer: determina el tamaño del buffer indicado.

Modos de apertura de archivo y su descripción:

modos	**Descripción**
r	Abre un archivo de sólo lectura. Este es el modo por defecto.
rb	Abre un archivo de sólo lectura en formato binario.
r+	Abre un archivo en modo de lectura y escritura.
rb+	Abre un archivo para lectura y escritura en formato binario.
w	Abre un archivo para sólo escritura. Sobrescribe el archivo si existe. Si no existe el archivo, crea un nuevo archivo para escritura.
wb	Abre un archivo para escribir en formato binario. Sobrescribe el archivo si

	existe. Si no existe el archivo, crea un nuevo archivo para escritura.
w+	Abre un archivo para escritura y lectura. Sobrescribe el archivo existente. Si no existe el archivo, crea un nuevo archivo para la lectura y la escritura.
WB+	Abre un archivo para escritura y lectura en formato binario. Sobrescribe el archivo existente. Si no existe el archivo, crea un nuevo archivo para la lectura y la escritura.
un	Abre un archivo para anexar. El puntero del archivo se encuentra al final si existe el archivo. Es decir, el archivo se encuentra en la modalidad de apertura. Si no existe el archivo, se crea un nuevo archivo para escritura.
ab	Abre un archivo para anexar en formato binario. El puntero del archivo se encuentra al final del archivo. Es decir, el archivo se encuentra en la modalidad de apertura. Si no existe el archivo, se crea un nuevo archivo para escritura.
a+	Abre un archivo tanto para anexar y lectura. El puntero del archivo se encuentra al final. El archivo se abre en la modalidad de apertura. Si no existe el archivo, se crea un nuevo archivo para la lectura y la escritura.
ab+	Abre un archivo tanto para anexar y la lectura en formato binario. El puntero del archivo se encuentra al final del archivo .El archivo se abre en la modalidad de apertura. Si no existe el archivo, se crea un nuevo archivo para la lectura y la escritura.

Atributos del objeto File

Una vez que se abre un archivo, se puede obtener diversa información relacionada con ese archivo. Aquí está una lista de algunos atributos relacionados con el objeto File:

Atributo	Descripción
file.closed	Devuelve true si el archivo está cerrado, falso en caso que este abierto.
file.mode	Devuelve el modo de acceso con el que se abrió el objeto.
file.name	Devuelve el nombre del archivo.

Ejemplo:

```
>>> archivo = open("prueba.txt", "wb")
>>> print "Nombre archivo: ", archivo.name
Nombre archivo:  prueba.txt
>>> print "Esta Cerrado?  : ", archivo.closed
Esta Cerrado?  :  False
>>> print "Modo de apertura : ", archivo.mode
Modo de apertura :  wb
>>>
```

El método close ()

El método close elimina cualquier información escrita en memoria y cierra el objeto File. Por otra parte un archivo se cierra automáticamente cuando el objeto de referencia de un archivo es reasignado a otro archivo. Es una buena práctica utilizar el método close () para cerrar un archivo.

```
>>> archivo = open("prueba.txt", "wb")
>>> print "Nombre archivo: ", archivo.name
Nombre archivo:  prueba.txt
>>> print "Esta cerrado ? " + str(archivo.closed)
Esta cerrado ? False
>>> print "Modo de apertura : ", archivo.mode
Modo de apertura :  wb
>>> archivo.close()
>>> print "Esta cerrado ? " + str(archivo.closed)
Esta cerrado ? True
>>>
```

Se puede apreciar en el código fuente como que después de ejecutar el método close, se cierra el puntero que estaba sobre el archivo.

Lectura y escritura de archivos

El objeto de File proporciona un conjunto de métodos de acceso para hacernos la vida más fácil. En los siguientes ejemplo vamos a utilizar los métodos read () y write () para leer y escribir archivos.

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

El método de write() no añade saltos de linea ('\ **n**') al final de la cadena, por lo que tendremos que agregarlo nosotros en el código:

Ejemplo:

```
>>> archivo = open("prueba.txt", "wb")
>>> archivo.write("www.python.org\n")
>>> archivo.write("www.google.com\n")
>>> archivo.write("www.gmail.com\n")
>>> archivo.close()
>>>
```

En el ejemplo agregamos 3 host al archivo prueba.txt

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

Ejemplo:

```
>>> archivo = open("prueba.txt", "r")
>>> contenido = archivo.read()
>>> print contenido
www.python.org
www.google.com
www.gmail.com

>>>
```

En la línea 1 abrimos en modo lectura (r) el archivo que ya estaba escrito por el ejemplo anterior y utilizamos el método read() para traernos el contenido.
Ahora si queremos leer línea por línea el archivo, python nos entrega el método readlines() que trasforma el contenido a un objeto list, lo que nos facilita leerlos con un ciclo for

Ejemplo:

```
>>> archivo = open("prueba.txt", "wb")
>>> archivo.write("www.python.org\n")
>>> archivo.write("www.google.com\n")
>>> archivo.write("www.gmail.com\n")
>>> archivo.close()
>>>
>>> archivo = open("prueba.txt", "r")
>>> for lista in archivo.readlines():
...     print lista
...
www.python.org

www.google.com

www.gmail.com

>>>
```

En Python desde la versión 2.5 adelante, nos entrega una forma más fácil de leer archivos con la palabra reservada **with**.

Ejemplo:

```
>>> with open("prueba.txt", "r") as archivo:
...     print archivo.read()
...
www.python.org
www.google.com
www.gmail.com

>>> archivo.closed
True
>>>
```

En el código fuente se puede apreciar que en pocas líneas de código leemos un objeto file y el mismo interprete se encarga de cerrar el archivo evitándonos utilizar el método close()

Manejo de Archivos JSON

13

Manejo de archivos JSON

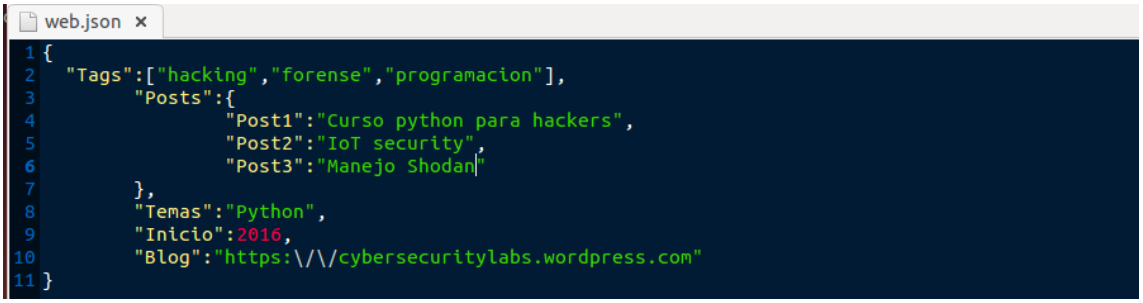
JSON, acrónimo de JavaScript Object Notation, es un formato de texto ligero para el intercambio de datos. JSON es un subconjunto de la notación literal de objetos de JavaScript aunque hoy, debido a su amplia adopción como alternativa a XML, se considera un formato de lenguaje independiente.

Instaleremos librería necesaria

pip install simplejson

Para comenzar con el trabajo de datos en formato JSON vamos a escribir en la Terminal

gedit web.json

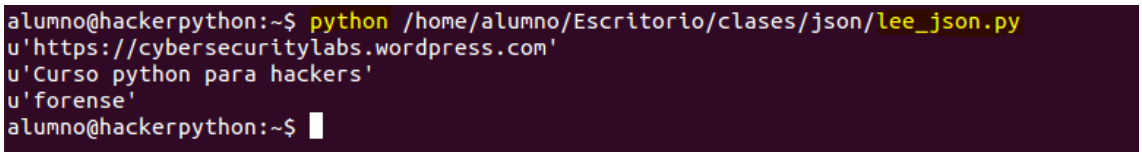


Ahora vamos a escribir un script que lea este objeto json

gedit lee_json.py

En el código fuente se puede apreciar que para trabajar con este tipo de objeto, se tiene que importar la librería json, además para acceder a los datos, se tiene que ir desde padre a hijo en forma descendiente.

Para acceder al resultado basta con escribir en la Terminal lo siguiente:



Manejo de archivos XML

14

Leer archivos formato XML

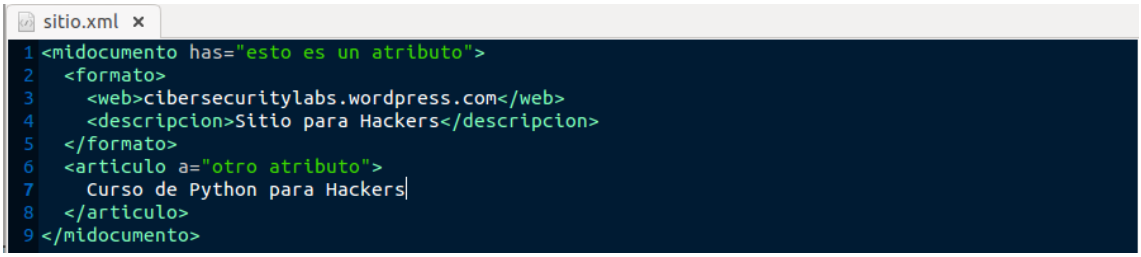
XML, siglas en inglés de eXtensible Markup Language ("lenguaje de marcas Extensible"), es un meta-lenguaje que permite definir lenguajes de marcas desarrollado por el World Wide Web Consortium (W3C) utilizado para almacenar datos en forma legible. A diferencia de otros lenguajes, XML da soporte a bases de datos, siendo útil cuando varias aplicaciones deben comunicarse entre sí o integrar información.

Instalaremos la librería necesaria

pip install xmltodict

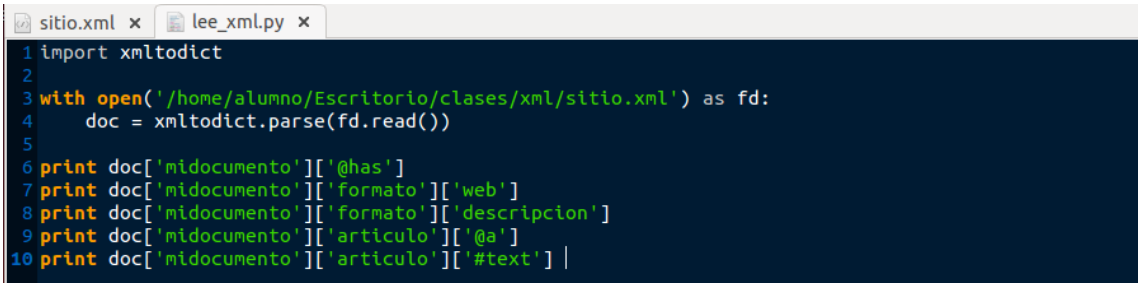
Para comenzar con el trabajo de datos en formato XML vamos a escribir en la Terminal

gedit sitio.xml



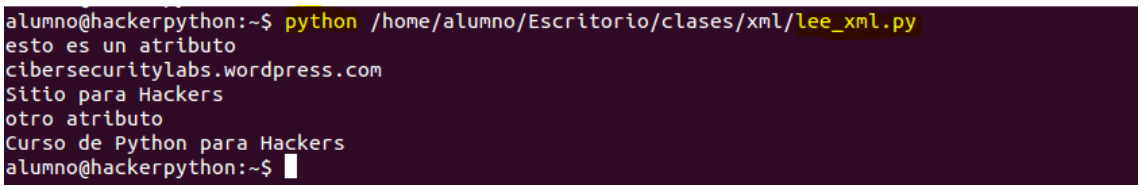
Ahora vamos a escribir un script que lea este objeto XML

gedit lee_xml.py



Para acceder a los datos se realiza en forma de árbol, de padre a hijo, en este ejemplo solo vemos hasta 3 niveles.

Si queremos ver los resultados vasta con escribir en la Terminal:



Librería OS

15

Liberia OS - Manejo de archivos y carpetas

Python nos permite acceder a funcionalidades del sistema operativo mediante la librería OS, esta librería permite manipular estructuras de directorios, leer, escribir, modificar y eliminar archivos.

Conocer directorio Actual

El método **getcwd()** (**c**urrent **w**orking **d**irectory), tal como lo indica su nombre, retorna la ruta actual desde donde se ejecuta nuestro script, o bien el directorio de instalación de Python en caso de tratarse del intérprete.

```
>>> import os
>>> os.getcwd()
'/home/alumno'
>>>
```

Listar archivos y carpetas

El método listdir (**List Dir**ectory) nos sirve cuando deseamos conocer que archivos y carpetas existen en un directorio específico.

```
>>> os.listdir('/var/')
['metrics', 'spool', 'backups', 'log', 'lib', 'opt', 'tmp', 'local', 'lock', 'ru
n', 'cache', 'mail', 'crash']
>>>
>>>
```

Como se puede apreciar en el código fuente al método se le pasa como parámetro la ruta que se desea consultar.

Por otra parte si se quiere consultar el directorio actual, tenemos que escribir el símbolo punto como argumento (.)

```
>>> os.listdir('.')
['.dbus', '.gconf', '.dmrc', 'examples.desktop', '.mozilla', '.bash_history', 'P
\xc3\xbablico', '.ICEauthority', '.pip', '.vboxclient-display.pid', '.xsession-e
rrors', '.Xauthority', 'Documentos', '.local', 'prueba.txt', 'Plantillas', 'repo
rte2.html', '.bashrc', '.cache', 'Descargas', 'M\xc3\xbasica', '.config', '.ninj
a_ide', '.profile', 'Im\xc3\xa1genes', 'Escritorio', '.vboxclient-seamless.pid',
 '.bash_logout', 'V\xc3\xaddeos', '.vboxclient-draganddrop.pid', '.vboxclient-cl
ipboard.pid', '.xsession-errors.old', 'datos.csv']
>>>
```

Creación de directorios

Para crear directorios usaremos el método mkdir (**Ma**k**e** **Dir**ectory) que como su nombre lo indica, crea un directorio. Este método necesita que se le pase como parámetro el nombre de directorio a crear.

```
>>> os.mkdir('informe')
>>> os.listdir('.')
['.dbus', '.gconf', '.dmrc', 'examples.desktop', '.mozilla', '.bash_history', 'P
\xc3\xbablico', '.ICEauthority', '.pip', '.vboxclient-display.pid', '.xsession-e
rrors', '.Xauthority', 'Documentos', '.local', 'prueba.txt', 'Plantillas', 'repo
rte2.html', '.bashrc', '.cache', 'Descargas', 'M\xc3\xbasica', '.config', 'infor
me', '.ninja_ide', '.profile', 'Im\xc3\xa1genes', 'Escritorio', '.vboxclient-sea
mless.pid', '.bash_logout', 'V\xc3\xaddeos', 'reportes', '.vboxclient-draganddro
p.pid', '.vboxclient-clipboard.pid', '.xsession-errors.old', 'datos.csv']
>>>
```

En el código se muestra que primero se crea un directorio y luego mediante el método listdir() se verifica su creación.

Por otra parte si se desea crear varias carpetas usaremos el método makedirs() de la siguiente manera.

```
>>> os.makedirs('documentacion/pentest/web')
>>> os.makedirs('documentacion/pentest/mobile')
>>> os.makedirs('documentacion/pentest/servidor')
>>> os.listdir('documentacion/pentest/')
['web', 'servidor', 'mobile']
>>>
```

Si queremos crear un directorio con permisos especiales, usaremos el segundo parámetro de el método mkdir() que si no se completa toma los permisos 777 por defecto.

```
>>>
>>> os.mkdir('carp_nueva',0o700)
>>>
```

Tabla permisos

Notación simbólica	Notación Numérica	Significado
----------	0000	Sin permisos
-rwx------	0700	leer, escribir, & ejecutar solo por el dueño
-rwxrwx---	0770	leer, escribir, & ejecutar por el dueño y el grupo
-rwxrwxrwx	0777	leer, escribir, & ejecutar por cualquira
---x--x--x	0111	ejecutar
--w--w--w-	0222	escribir
--wx-wx-wx	0333	escribir & ejecutar
-r--r--r--	0444	leer
-r-xr-xr-x	0555	leer & ejecutar
-rw-rw-rw-	0666	leer & escribir
-rwxr-----	0740	El dueño puede leer, escribir, & ejecutar; el grupo solo puede leer; otros usuarios no tienen permisos

Eliminar directorios y archivos

Para eliminar directorios Python nos entrega el método rmdir(**Rem**ove **Dir**ectory) para eliminar un directorio o removedirs() para eliminar directorios y remove() para eliminar archivos.

```
>>> os.rmdir('informe')
>>> os.removedirs('documentacion/pentest/mobile')
>>> os.remove('documentacion/pentest/web/reporte.html')
>>>
```

Recorriendo un directorio

Ahora vamos a poner en práctica lo aprendido con un pequeño ejemplo.

```
>>> import os
>>> root_dir = "/home/alumno/Escritorio/clases"
>>> for dirnombre, subdirectorio, listaarchivo in os.walk(root_dir):
...     print "Nombre Directorio: " + dirnombre
...     for archivo in listaarchivo:
...         print "Archivo : " + archivo
...
Nombre Directorio: /home/alumno/Escritorio/clases
Archivo : importador.py~
Nombre Directorio: /home/alumno/Escritorio/clases/clase
Archivo : importador.py
Archivo : fbruta.py
Nombre Directorio: /home/alumno/Escritorio/clases/json
Archivo : web.json
Archivo : lee_json.py
Archivo : lee_json.py~
Archivo : usuarios.json~
Nombre Directorio: /home/alumno/Escritorio/clases/xml
Archivo : lee_xml.py~
Archivo : sitio.xml
Archivo : sitio.xml~
Archivo : lee_xml.py
>>>
```

Línea 1: Se importa la librería os.
Línea 2: Se establece la ruta que se quiere recorrer.
Línea 3: Se recorren los directorios con el ciclo for y el resultado se guardan en las variables dirnombre, subdirectorio, listaarchivo.
Línea 4: Se imprime la carpeta actual que se esta recorriendo.
Línea 5: Se crea un ciclo for para recorrer los archivos que se encuentren en la carpeta.
Línea 6: Se imprime el nombre del archivo.