26. **Python.**

26.1 Tcl/Tk, un primer ejemplo de lenguaje de script.

Tcl/Tk es un lenguaje de script que puede ser utilizado para las fases de prototipado y prueba, realización de aplicaciones, etc. También nos permite la creación de programas dotados de interfaces gráficas GUI. Puedes encontrar un magnífico tutorial en ***http://www6.uniovi.es/tcl/tutorial/***.

Como pequeño ejemplo de iniciación, observa el siguiente código que muestra una ventana con el típico Hola Mundo:

```
#! /usr/bin/wish
pack [button .b -text "Hola Mundo" -command {exit}] -padx 30 -pady 20
```

Si quieres algo mejor puedes utilizar TKCon, en Ubuntu instala el paquete *tkcon* y tendrás una consola con más características (histórico de órdenes, configurable, múltiples consolas cada una con su estado independiente, posibilidad de cortar/copiar/pegar entre ventanas, etc.) que la consola normal para la programación en TCL/TK, consulta la página Web **http://tkcon.sourceforge.net/** para más información.

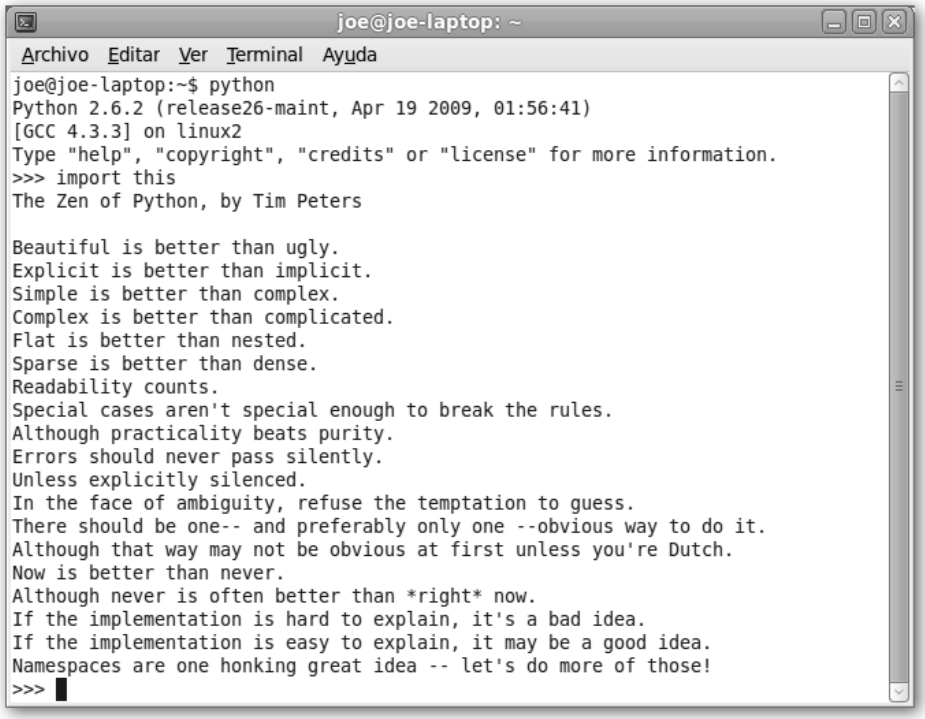
26.2. ¿Qué es Python?

Python es un lenguaje de programación moderno, cuyo nombre no deriva de serpiente sino de los famosos humoristas Monty Python. La primera versión salió a la luz en 1991 y entre sus características destacan:

- Lenguaje de *propósito general*, sin embargo quizás no es una buena opción para programas intensivos en cómputo o de bajo nivel. Para esta idea te recomendamos mejor C.
- *Orientado a objetos*.
- *Lenguaje de script* (tipo Tcl, Perl, Ruby, etc.) de alto nivel.
- *Multiplataforma*, corre en MacOS, Windows y Linux.
- Destaca por ser *fácil de aprender* y que el código que se escribe suele ser bastante *legible*. Para empezar a conocer Python aparte de los siguientes capítulos, un lugar de peregrinación obligada es ***http://docs.python.org/***, y más específicamente las secciones: Tutorial, Library Reference y Python HOWTOs. También puedes leer online gratuitamente Dive Into Python,

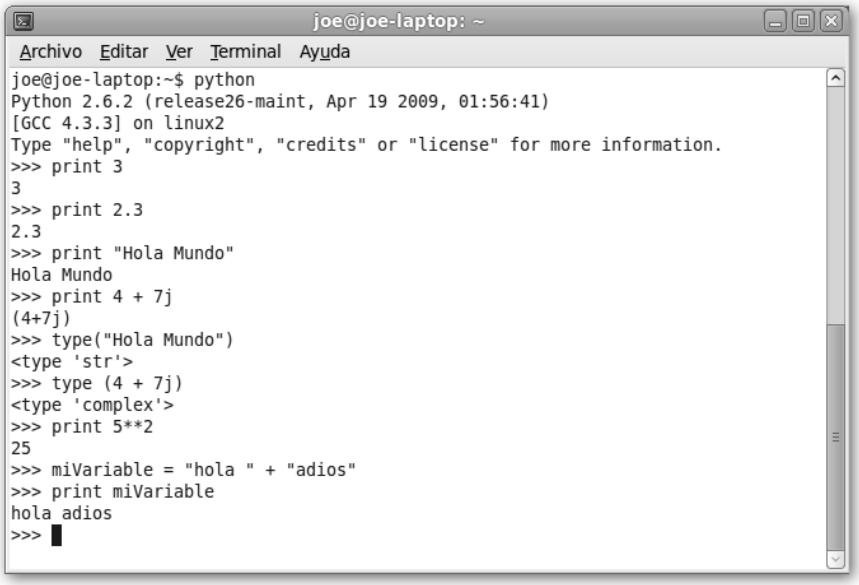
http://diveintopython.org/index.html, Learning To Program, ***http://www.freenetpages.co.uk/hp/alan.gauld/***, y How to Think Like a Computer Scientist, ***http://openbookproject.net//thinkCSpy/***.

- Suele llamar la atención el poco uso de signos de puntuación tan típicos de otros lenguajes como el punto y coma ";". Python en cambio destaca por el uso de identado para definir los bloques y los dos puntos ":".
- Python es *libre*, *open source* y presenta una comunidad de usuarios muy fuerte que lo apoya.
- ¿Y el "Hola Mundo"? Python muestra en vez del típico "Hola Mundo", su Zen, que se trata de una declaración de intenciones: simple es mejor que complejo, la legibilidad es importante, si la implementación es difícil de explicar es una mala idea, etc. Escribe en la consola *python* y luego *import this*.



26.3. Sintaxis básica de Python.

Python puede ser utilizando directamente en modo intérprete interactivo (basta con escribir python en la consola de Linux) o mediante ficheros con extensión py, luego se ejecutan escribiendo en la consola *python nombreFuente.py*.

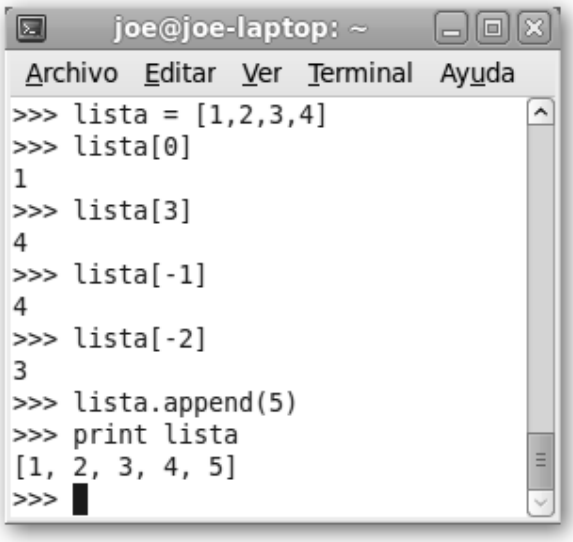


Como puedes ver en la figura Python dispone de los siguientes tipos:

- Tipos numéricos: enteros (3), flotantes (4.0), largos (4L). ¡También complejos! Observa *4 + 7j*, donde j representa la i, imaginario.
-

- Cadenas: En comillas simples o dobles, por ejemplo “Hola Mundo”. Algunos métodos de cadenas son:
  1. len, longitud (p.e. *len(‘Hola Mundo’)*, será 10.
  2. *“Hola Mundo”.find(‘Mundo’)*, devuelve 5, pues el carácter ‘H’ está en la posición 0, ‘o’ en la 1, ‘l’ (2), ‘a’ (3), “ “ (4), y la ‘M’ de ‘Mundo’ en la 5.
  3. *“Hola Mundo”.replace(‘Mundo’, ‘Python’)*, resulta en ‘Hola Python’.
  4. Si quieres pasar la cadena a mayúsculas: *“Hola Mundo”.upper()* .
  5. *“Hola Python, Python, Python”.count(‘Python’)* devuelve el número de ocurrencias de la subcadena Python, en este caso tres. Observa que en los ejemplos hemos utilizado indistintamente comillas simples o dobles.

- Listas: *lista = [1, 2, 3, 4, 5]*. Las tuplas son muy similares, sólo que no pueden modificarse. Se definen con *miTupla = (1, 2, 3)*, si quieres verla bastará con escribir en la consola *print miTupla* (mostrará (1,2,3)) y su tipo se solicita mediante *type(miTupla)* que devolverá <type ‘tuple’>.



Observa que hemos definido la lista. Si la lista tiene n elementos, se puede iterar desde 0 a n-1.

Llama la atención que **-1** hace referencia al último elemento de la lista. Además, observa el uso de **append** para añadir un nuevo elemento en la cola de la lista, también podemos utilizar *lista.insert(posición, elemento)*. Finalmente, con el operador +, concatenamos listas.

- Los diccionarios son conjuntos de pares: clave, valor. Para acceder a un diccionario debes recordar que *los índices son las claves*. Por ejemplo, un diccionario sería: *misTelefonos = {'Marta':952243212, 'Javier':956123456, 'Pedro':951231234}*, para acceder al teléfono de Pedro bastaría con escribir *print misTelefonos['Pedro']*. Para comprobar si una clave está en el diccionario tenemos *in*, por ejemplo *'Marta' in misTelefonos* devuelve True.

Instalemos para empezar el entorno de desarrollo IDLE, basta con instalar el paquete ***idle*** y se encontrará en **Aplicaciones**, **Programación**, **IDLE**. Una vez lanzado selecciona **File**, **New Window** para abrir un editor donde podremos escribir nuestro código Python, salvarlo con **File**, **Save** o **Ctrl**, **S** y cargar nuestro fichero fuente en la Shell de Python mediante **Run**, **Run Module** o **F5**.

La idea será crear un módulo Python llamado myPrimer.py e implementaremos tres funciones en él. Más tarde podremos utilizar dichas funciones desde el Shell de Python (ventana de la derecha) sin más que cargar el módulo con **F5** tal como se comentó anteriormente. *Debes prestar especial atención al identado pues será el que determine el ámbito de las funciones y de los bloques* asociados al código a ejecutar si la condición es cierta o falsa. Veamos en primer lugar una comparación entre C y Python:

C *if (condición){* *Bloque1;* *} else {* *Bloque2;* *}*	*Python:* *if condición:* *Bloque1* *else:* *Bloque2*

A continuación, veamos un if anidado en Python:

```
if condición:
        Bloque1
elif condición2:
        Bloque2
else:
        Bloque3
```

```
while  condición:
        Bloque
#Se ejecuta Bloque mientras se satisfaga condición.
for elemento in lista:
        Bloque
#Se ejecuta Bloque tantas veces como elementos en la lista. Observa que elemento va tomando como valor los diferentes elementos en la lista.
```

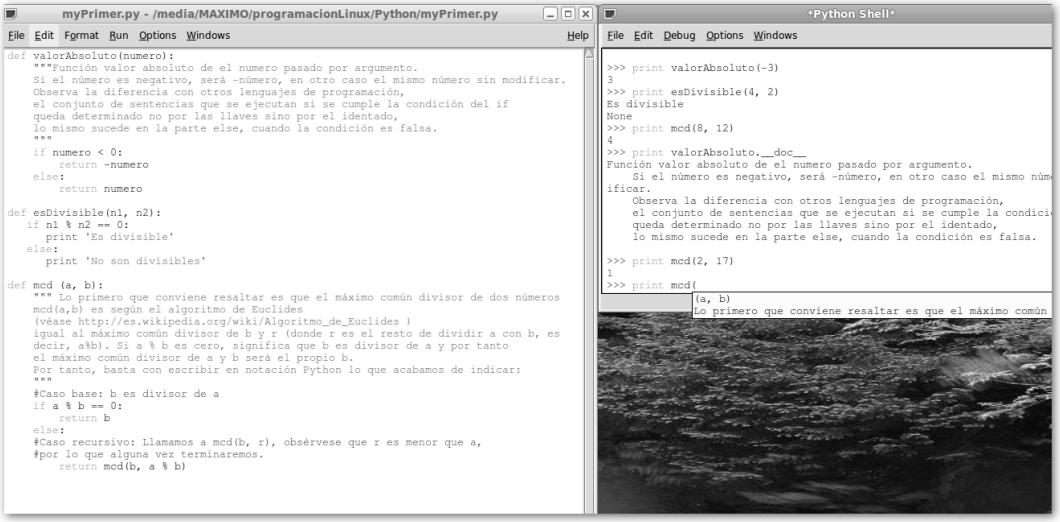
Finalmente, mostramos cómo se implementan los *bucles* en Python:

```
def muestraTablaAscci():
;;; Vamos a mostrar la tabla ASCII haciendo uso de la estructura for que hace que n itere desde 0 a 256-1 es decir 255.
;;;
    cadena = ''
    for n in range(0, 256):
        cadena = cadena + ' ' + chr(n)

    return cadena
```

Fíjate también que los comentarios son una almohadilla # para una línea y para comentarios multilíneas empiezan y acaban con tres comillas dobles “”””. Los comentarios detrás de la declaración de la función def nombreFunción(lista argumentos) se interpretan como documentación y podrán ser llamadas mediante nombreFunción.__doc__ donde dos guiones bajos “_ _” están delante y detrás de la palabra doc (documentación).

En general no es una buena idea llamar directamente a estos métodos que presentan dos guiones (__init__, __add__, etc.), será mejor dejar que Python los invoque automáticamente.



Conviene resaltar que en ningún momento hemos definido el tipo de ningún parámetro, variable, etc. ¡Python realizará esta tarea por nosotros! Veamos ahora el código más detenidamente.

```
def valorAbsoluto(numero):
“””La función valor absoluto de un número es bastante sencilla. Si el número
es negativo, será –número, en otro caso, el mismo número sin modificar.
Observa la diferencia con otros lenguajes de programación, el conjunto de
sentencias que se ejecutan si se cumple la condición del if queda determinado
no por las llaves sino por el identado, lo mismo sucede en la parte else,
cuando la condición es falsa. “””
    if numero < 0:
        return –numero
    else:
        return numero

def esDivisible(n1, n2):
#Python dispone de los operadores habituales :+, -, *, /, % (resto entero)
    if n1 % n2 == 0:
        print 'Es divisible'
    else:
        print 'No son divisibles’

def mcd (a, b):
“””Lo primero que conviene resaltar es que el máximo común divisor de dos
números mcd(a,b) es según el algoritmo de Euclides (véase
http://es.wikipedia.org/wiki/Algoritmo_de_Euclides) igual al máximo común
divisor de b y r (donde r es el resto de dividir a con b, es decir, a%b). Si a % b
es cero, significa que b es divisor de a y por tanto el máximo común divisor
de a y b será el propio b. Por tanto, basta con escribir en notación Python lo
que acabamos de indicar:“””
#Caso base: b es divisor de a
            if a % b == 0:
                        return b
            else:
#Caso recursivo: Llamamos a mcd(b, r), obsérvese que r es menor que a, por
lo que alguna vez terminaremos.
                        return mcd(b, a % b)
```

26.4. Clases en Python.

Creemos nuestra primera clase con Python. Vamos a realizar una versión de la implementación de la clase Fracción que puedes encontrar con algunas pequeñas variaciones en ***http://openbookproject.net/thinkcs/python/english/app02.html***. Para crear nuestra clase Fracción, basta con escribir *class NombreDeClase:*, a partir de ahí todo lo que esté dentro del alcance de la identación de la clase, Python considerará que es código relativo a la clase.

```
class Fraccion:
"""Este método nunca deberá ser llamado directamente, de ahí que tenga esta forma tan extraña. Este método se invocará cuando creemos una instancia de la clase. Observa que todos los métodos tienen como primer argumentos self, el segundo y el tercer argumento de init son el numerador y denominador respectivamente. También considera que el usuario no precisa pasar una referencia self y que el argumento denominador toma por defecto el valor 1, si al crear la instancia sólo se pasa un parámetro.
"""
        def __init__(self, numerador, denominador=1):
"""Si el usuario trata de crear una instancia de la clase Fracción con los argumentos 8 y 12, nosotros vamos a almacenarlo en la forma simplificada. Esto es sólo un ejemplo de implementación, existen más variantes como hemos comentado al principio. Así como mcd(8,12)=4, la fracción tal como va a ser almacenada resultará en numerador = 8/4 =2 y denominador 12/4=3, es decir, en este caso, nuestra fracción será 2/3."""
                self.numerador = numerador / mcd(numerador,denominador)
                self.denominador = denominador / mcd(numerador,denominador)

"""Al implementar el método __str__, con dos guiones bajos tanto delante como detrás de str (string, cadena) modificamos la forma en que se visualizará nuestra Fracción. Este método tampoco lo llamaremos directamente, será Python quien lo invoque cuando llamemos a print sobre una fracción."""
        def __str__(self):
#Si el denominador es 1, mostramos simplemente el denominador, y evitamos mostrar una fracción como 5/1, que queda francamente mal.
        if (self.denominador == 1):
                return "%d" % self.numerador
#En el caso de que el denominador no sea uno, lo mostramos de la forma en que esperamos se visualice normalmente una fracción, por ejemplo: 6/5.
        return "%d/%d" % (self.numerador, self.denominador)


"""El siguiente método que vamos a modificar es __add__, nos permitirá sumar fracciones tal como cualquier alumno de ESO espera, 4/5+3/4."""
        def __add__(self, otraFraccion):
#Si el tipo de otraFraccion es el mismo que el número 7, es decir, un entero, lo convertimos a una instancia de Fracción. ¿Por qué 7? Porque es nuestro número favorito…
        if type(otraFraccion) == type(7):
                otraFraccion = Fraccion(otraFraccion)
#Recuerda que a/b + c/d = ad/bd + cb/bd = (ad+cb)/bd.
        return Fraccion(self.numerador * otraFraccion.denominador + self.denominador * otraFraccion.numerador, self.denominador * otraFraccion.denominador)
```

#Tal como está escrito funcionara para 3/5 + 4/7 o para 3/5 + 3 donde 3 es un entero, pero,… ¿qué sucedería con 3 + 3/5?, que no funcionaría, de ahí que también implementemos __radd__, en este caso Python interpretará que 3 es el parámetro.

```
__radd__ = __add__
```

#Realizamos también la resta de fracciones:

```
def __sub__(self, otraFraccion):
    if type(otraFraccion) == type(7):
        otraFraccion = Fraccion(otraFraccion)
#Recuerda que a/b - c/d = ad/bd – cb/bd = (ad – cb)/bd.
    return Fraccion(self.numerador * otraFraccion.denominador - self.denominador * otraFraccion.numerador, self.denominador * otraFraccion.denominador)
```

#Considera que no podemos llamar a rsub porque 3 – 3/5 no es lo mismo que 3/5 – 3, es decir, **la resta de fracciones no es conmutativa**, lo mismo sucederá con la división.

#La única novedad es reconocer que a/b * c/d = a c / bd.

```
def __mul__(self, otraFraccion):
    if type(otraFraccion) == type(7):
        otraFraccion = Fraccion(otraFraccion)
    return Fraccion(self.numerador * otraFraccion.numerador, self.denominador * otraFraccion.denominador)

__rmul__ = __add__
```

#Finalmente también podemos dividir fracciones sin más que recordar que a/b / c/d = a/b * d/c = ad/bc.

```
def __div__(self, otraFraccion):
    if type(otraFraccion) == type(7):
        otraFraccion = Fraccion(otraFraccion)
    return Fraccion(self.numerador * otraFraccion.denominador, self.denominador * otraFraccion.numerador)
```

“””Todos los módulos en Python son objetos y como tales tienen atributos, en particular tiene el atributo __name__. Si el módulo es importado, el nombre del módulo será el nombre del fichero del módulo sin ruta ni extensión, es decir, en este caso miFraccion. Si el módulo es utilizado como programa standalone, independiente, entonces esta propiedad adquiere el valor __main__, lo que aprovechamos para crear y mostrar una instancia de Fraccion, más concretamente 3/5. Este se llamará como python myFraccion.py.”””

```
if __name__ == '__main__':
    a = Fraccion(3,5)
    print "Mi Fraccion de prueba: ", a
```

Veamos un fichero Python que importa nuestro módulo miFraccion.py, lo llamaremos usaFraccion.py:

```
#python usaFraccion.py
from miFraccion import Fraccion

#Creamos dos instancias a y b, las sumamos y restamos.
a = Fraccion(6,5)
b = Fraccion(4,5)
print a + b
# 6/5+4/5=10/5=2.
print a – b
#6/5-4/5=2/5.
#Ahora creamos otras dos instancias, c y d, probamos también las operaciones multiplicación y división.
c = Fraccion(2,5)
d = Fraccion(5,2)
print c * d
#2/5*5/2=2*5 / 5*2=10/10 = 1
print c / d
#2/5 / 5/2 = 2/5 * 2/5 = 2*2/5*5 = 4/25.
print Fraccion(1,2) / 2
#1/2 / 2 = ½ / 2/1 = ½ * ½ = 1*1 / 2*2 = ¼.
```

En la figura inferior se muestra un ejemplo de ejecución directamente en la consola, basta escribir *python usaFraccion.py*:

26.5. GUI con wxPython.

Para desarrollar interfaces gráficas de usuario Python cuenta con dos soluciones principales: PyGTK y wxPython. Buenos manuales de ambas opciones los puedes encontrar en: ***http://www.pygtk.org/pygtk2tutorial-es/index.html*** y ***http://zetcode.com/wxpython/*** respectivamente.

Vamos a mostrar wxPython, para lo que precisamos instalar el paquete **python-wxgtk2.8**, luego creamos el fichero miPrimerWxgtk.py

```
#!/usr/bin/python

import wx
# Importamos los módulos wxPython
miApp = wx.App()

"""Todo programa en wxPython precisa tener un objeto aplicación, además creamos un objeto Frame, que será nuestro formulario principal. El constructor indica que se trata del Widget raíz, es decir, no tiene padres (None), con -1 indicamos a Python que le asigne un identificador y finalmente el tercer parámetro será el título de nuestra ventana.
miFrame = wx.Frame(None, -1, 'Hola WxPython')."""

# Con el método Show solicitamos que se muestre nuestra ventana.
miFrame.Show()

""" Invocamos al método MainLoop de nuestro objeto aplicación para entrar en el típico bucle de cualquier aplicación de ventanas en un entorno GUI donde se dispararán eventos y habrá manejadores asignados para darles respuesta."""
miApp.MainLoop()
```

El resultado no es muy espectacular, pero tenemos una ventana solvente con los típicos botones de minimizar, maximizar y cerrar:

Veamos ahora un ejemplo algo más elaborado:

```
#!/usr/bin/python
import wx
import wx.html

#Creamos una cadena que contiene el código HTML de una página.
miPaginaHtml = '<html><body><H1>Hola Python</H1> \
</body></html>'

#Nuestra clase heredará de wx.Frame.
class HolaWxPython2(wx.Frame):

#El constructor invoca a la clase padre, con la única diferencia respecto
al ejemplo anterior de haber asignado también el tamaño de la ventana.
    def __init__(self, parent, identificador, titulo):
        wx.Frame.__init__(self, parent, identificador, titulo, size=(350,
250))

#Creamos un objeto barra de menú.
        barraMenu = wx.MenuBar()

#Dicha barra tendrá un primer menú fichero con un submenú salir.
        fichero = wx.Menu()
        salir = wx.MenuItem(fichero, 1, '&Salir\tCtrl+Q')
#El submenú salir se crea con tres parámetros: fichero (del que es
submenú), un identificador y el texto que se mostrará en la barra de
menú (observa cómo definimos las teclas rápidas).

        salir.SetBitmap(wx.Bitmap('imagenes/exit.png'))
#Incluimos una imagen wx.Bitmap asociado a dicho submenú

        fichero.AppendItem(salir)
#Añadimos el ítem o submenú salir a fichero.

        barraMenu.Append(fichero, '&Fichero')
#Incluimos el menú fichero en la barra de menú.

        self.SetMenuBar(barraMenu)
#Y añadimos nuestra barra de menú al frame o ventana principal.
Obsérvese que se trata de unos widgets que están contenidos unos en
otros, de menor a mayor sería: salir, fichero, barraMenu,
self(wx.Frame).

        self.Bind(wx.EVT_MENU, self.OnSalir, id=1)
#Dotamos de funcionalidad a nuestra barra de menú, la línea anterior
indica que el evento de menú hacer clic en el widget con identificador
uno, es decir, salir (submenú de fichero), será tratado por el manejador
OnSalir.
```

#Creamos un panel de color rojo y una caja horizontal ¡Ojo!, ya sabemos que el diseño no es nuestro fuerte…

miPanel = wx.Panel(self, -1)
miPanel.SetBackgroundColour(wx.RED)
miCajaHorizontal = wx.BoxSizer(wx.HORIZONTAL)

#El Widget wx.html.HtmlWindow permite visualizar páginas Web, indicamos que se encuentra en el panel miPanel y que no presenta bordes, así como con el método *SetPage* la página HTML que queremos visualizar.

miHtmlWindow = wx.html.HtmlWindow(miPanel, -1, style=wx.NO_BORDER)

#Aquí hay un detalle que conviene aclarar, miHtmlWindow tiene como padre a miPanel, porque un panel es una ventana pero no ocurre así con la caja wx.Box que hereda de wx.Size, el cual a su vez hereda de wx.Object. En esencia, *una caja es un contenedor que no es visible y que por tanto no hereda de wx.Windows*.

miHtmlWindow.SetPage(miPaginaHtml)

El control miHtmlWindow vamos a ponerlo de color CYAN para que veas bien el borde que definimos.

miHtmlWindow.SetBackgroundColour(wx.CYAN)

#Agregamos el control HtmlWindow a la caja horizontal, con wx.ALL indicamos que se añada un borde de 20 en las cuatro caras, con wx.EXPAND solicitamos que miHtmlWindow ocupe todo el espacio disponible.

miCajaHorizontal.Add(miHtmlWindow, -1, wx.EXPAND | wx.ALL, 20)

#Indicamos que la configuración de la posición de los distintos Widgets en nuestro panel está determinada por el objeto miCajaHorizontal. Hemos utilizado una caja wx.BoxSizer, otras opciones disponibles son: wx.StaticBoxSizer, wx.GridBagSizer, wx.FlexGridSizer, wx.GridSizer.

miPanel.SetSizer(miCajaHorizontal)

#Finalmente, indicamos a la ventana que se posicione en el centro y la mostramos.

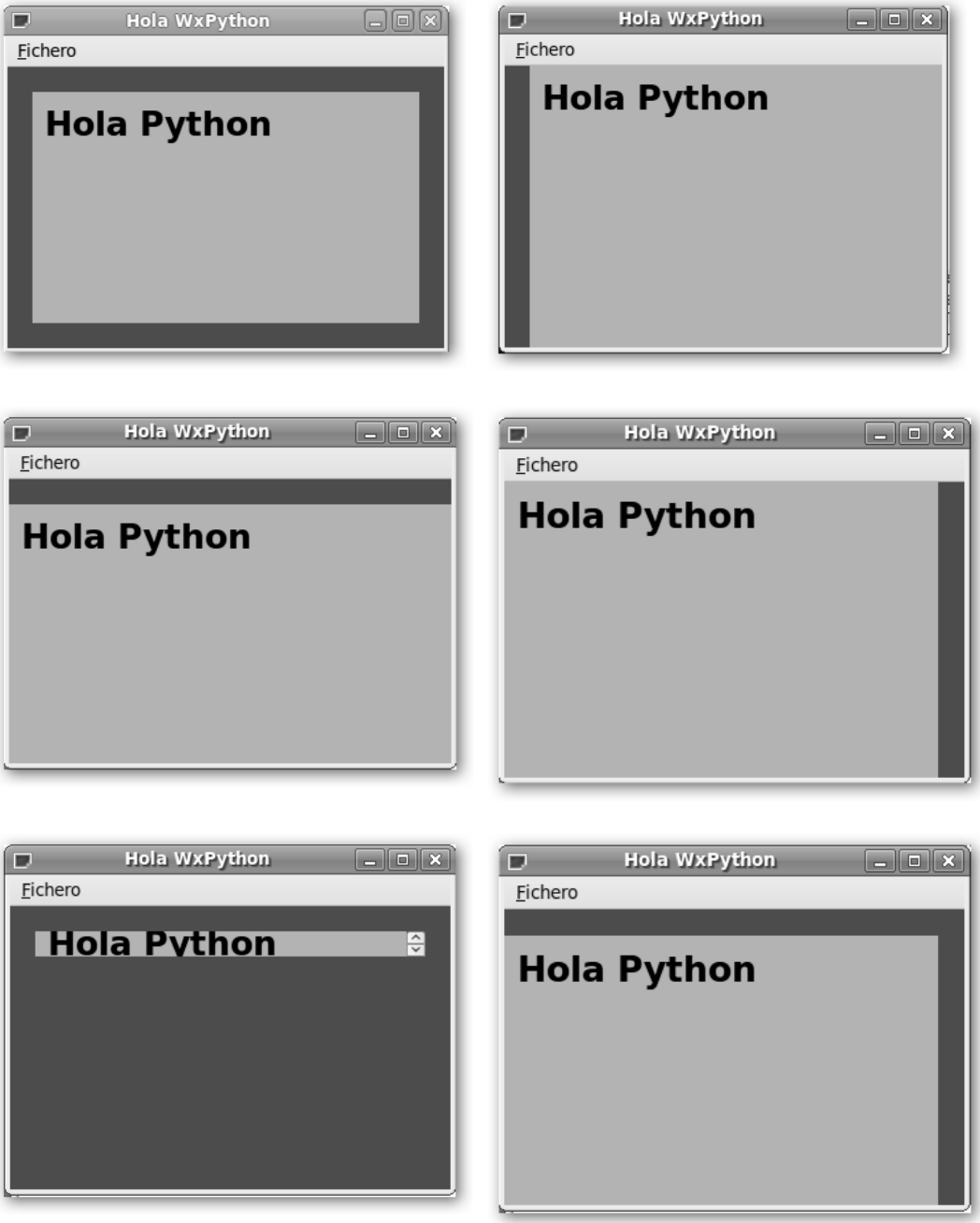
self.Centre()
self.Show(True)

#El manejador del evento hacer clic en **Salir** es muy fácil, provoca que la ventana se cierre.

```
def OnSalir(self, event):
    self.Close()

miApp = wx.App()
HolaWxPython2(None, -1, 'Hola WxPython')
miApp.MainLoop()
```

Como puedes observar con muy pocas líneas hemos conseguido una aplicación que nos permite mostrar páginas HTML. En la primera figura se muestra lo obtenido con la configuración *miCajaHorizontal.Add(miHtmlWindow, -1, wx.EXPAND | wx.ALL, 20),* en las siguientes figuras se observa lo que resulta al sustituir *wx.ALL* por *wx.LEFT, wx.TOP, wx.RIGHT, wx.BOTTOM y wx.TOP | wx.RIGHT:*



En la siguiente versión vamos a incorporar en este ejemplo de introducción a wxPython el submenú **Acerca** para mostrar la típica ventana de diálogo Acerca de la aplicación.

```
#!/usr/bin/python
import wx
import wx.html

class HolaWxPython3(wx.Frame):
    def __init__(self, parent, identificador, titulo):
        wx.Frame.__init__(self, parent, identificador, titulo, size=(350, 250))

        barraMenu = wx.MenuBar()
        fichero = wx.Menu()
        salir = wx.MenuItem(fichero, 1, '&Salir\tCtrl+Q')
        salir.SetBitmap(wx.Bitmap('imagenes/exit.png'))
        fichero.AppendItem(salir)
        barraMenu.Append(fichero, '&Fichero')
```

#Vamos a crear otra entrada de menú **Ayuda** del que dependerá un submenú **Acerca de**.

```
        ayuda = wx.Menu()
        acerca = wx.MenuItem(fichero, 2, 'Acerca de')
        acerca.SetBitmap(wx.Bitmap('imagenes/acerca.png'))
        ayuda.AppendItem(acerca)
        barraMenu.Append(ayuda, 'A&yuda')
        self.SetMenuBar(barraMenu)

        self.Bind(wx.EVT_MENU, self.OnSalir, id=1)
```

#Indicamos que el manejador OnAcerca responderá al evento hacer clic en el Widget con identificador 2, acerca.

```
        self.Bind(wx.EVT_MENU, self.OnAcerca, id=2)

        miPanel = wx.Panel(self, -1)
        miCajaHorizontal = wx.BoxSizer(wx.HORIZONTAL)
        miHtmlWindow = wx.html.HtmlWindow(miPanel, -1, style=wx.NO_BORDER)
```

#En esta ocasión vamos a cargar una dirección de una url remota con el método *LoadPage*.

```
        miHtmlWindow.
LoadPage("http://docs.python.org/tutorial/index.html")
        miCajaHorizontal.Add(miHtmlWindow, -1, wx.EXPAND | wx.ALL, 10)
        miPanel.SetSizer(miCajaHorizontal)
        self.Centre()
        self.Show(True)
```

#El manejador **OnAcerca** mostrará una ventana de diálogo Acerca de que está implementada por la clase wx.AboutBox al cual se le pasa un objeto de la clase wx.AboutDialogInfo con toda la información relativa

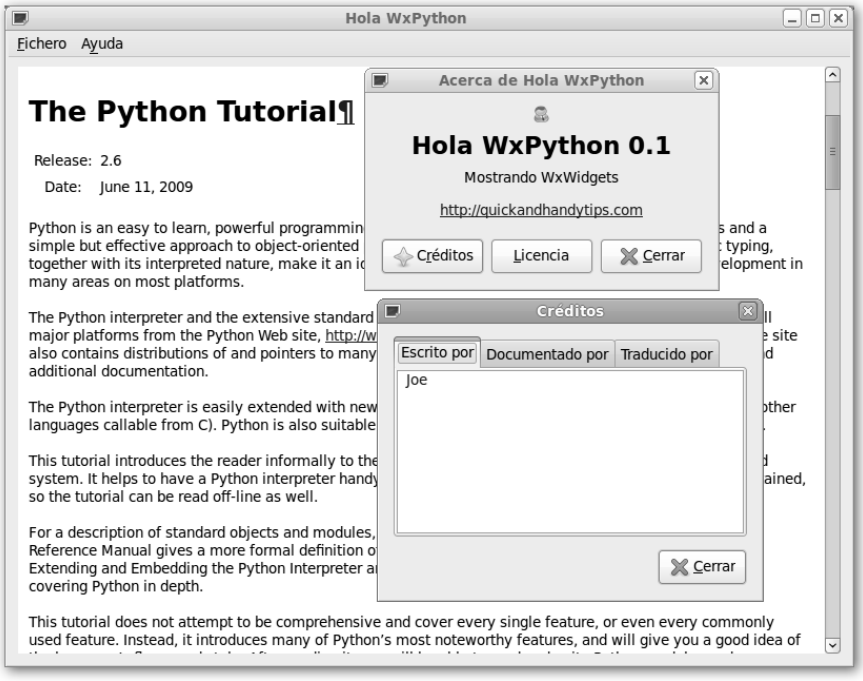
a nuestra aplicación, en particular: icono, nombre, versión, descripción, desarrollador, traductor, documentadores, licencia y página Web por orden de presencia en el código:

```
    def OnAcerca(self, event):
        miInfo = wx.AboutDialogInfo()
        miInfo.SetIcon(wx.Icon('imagenes/acerca.png',
wx.BITMAP_TYPE_PNG))
        miInfo.SetName('Hola WxPython')
        miInfo.SetVersion('0.1')
        miInfo.SetDescription('Mostrando WxWidgets')
        miInfo.AddDeveloper('Joe')
        miInfo.AddTranslator('John Smith')
        miInfo.AddDocWriter('Marta Nunez')
        miInfo.SetLicence('GPLv3')
        miInfo.SetWebSite('http://quickandhandytips.com')
        wx.AboutBox(miInfo)

    def OnSalir(self, event):
        self.Close()

app = wx.App()
HolaWxPython3(None, -1, 'Hola WxPython')
app.MainLoop()
```

Observa nuestra ventana principal y la ventana AboutBox superpuesta después de haber seleccionado **Ayuda**, **Acerca de**.



Finalmente, realicemos un ejercicio distinto con una aplicación que dado un número muestra bastante información sobre él.

```
#!/usr/bin/python
import wx
import wx.html

# Vamos a crear una lista de cadenas de texto.
miRepresentacionNumero = ['Cero', 'Uno\n*\nI','dos\n * *\nII', 'Tres\n
* * *\nIII', 'Cuatro\n * * * *\nIV', 'Cinco\n * * * * *\nVI', 'Seis\n * * * *
*  *\nVI', 'Siete \n*******\nVII', 'Ocho\n ********\nVIII',
'Nueve\n*********\nIX',                    'Diez\n**********\nX',
'Once\n**********\n*\nXI',                'Doce\n**********\n**\nXII',
'Trece\n**********\n***\nXIII',  'Catorce\n**********\n****\nXIV',
'Quince\n**********\n*****\nXV',
'Dieciseis\n**********\n******\nXVI',
'Diecisiete\n**********\n*******\nXVII',
'Dieciocho\n**********\n********\nXVIII',
'Diecinueve\n**********\n*********\nXIX',
'Veinte\n**********\n**********\nXX']

miPaginaHtml = '<html><body > \
</body>Hola Python</table></html>'

#El método divisor devuelve una cadena de texto con los divisores del
número que se le pasa como argumento.
def divisor(n):
    miCadena = '1'
    if n == 1:
        return miCadena
#El iterador i toma valores entre 2 y n/2. Por supuesto, no puede haber
ningún divisor de n mayor que n/2 a excepción del propio n.
    for i in xrange(2,n/2+1):
#Si i es divisor de n agregamos i a miCadena precedido de un espacio
en blanco.
        if n%i == 0: miCadena = miCadena + ' ' + `i`
    return miCadena + ' ' + `n`

#Devuelve una cadena indicando si el número que se le pasa como
parámetro es primo o no. La idea es que todo número es primo excepto
que exista un número entre 2 … n/2 que sea divisor suyo.
def primo(n):
    miCadena = 'Es primo'
    for i in xrange(2,n/2+1):
        if n%i == 0:
            miCadena = 'No es primo'

    return miCadena

#Mostramos una solución no recursiva al cálculo del factorial de un
número n.
```

```
def factorial(n):
    if n < 2:
            return \nFactorial: ' + '1'

    resultado = 1
#n!=1*2*....*n
    for i in xrange(2,n+1):
        resultado *= i

    return \nFactorial: ' + str(resultado)

#Convierte una cadena en un número.
def convierte(cadena):
    numero = 0
    i = -1
#Inicializamos numero a cero e "i", que indicará la posición del nuevo digito a -1.
    for c in cadena:
        i += 1
#Por cada uno de los caracteres en cadena, lo pasamos a entero con ord(c)-48 y lo sumamos a número según su valor. Existen dos elementos que indica el valor de un dígito en un número: su valor y su posición. El peso de la posición está indicado en (10**i), es decir 10^i.
        numero = (ord(c)-48) + numero * (10**i)

    return numero

#Mostramos de cada número su representación hexadecimal y binaria, es decir, en base 16 (0,1,…9,A,..F) y 2 (0,1). Además, si el número es menor que 21 mostramos un texto con la representación del número en lenguaje natural y romano (se deja al lector como ejercicio hacerlo con una función).
def representacion(numero):
    hexadecimal = "%X" % numero
    cadena = \nRepresentacion Hexadecimal: ' + hexadecimal + \nRepresentacion Binaria: ' + decimalABinario(numero)

    if (numero<21):
            return \n' + miRepresentacionNumero[numero] + cadena + \n'

    return cadena

#Convierte un número de decimal a binario.
def decimalABinario(numero):
    if (numero < 0):
        return 'Imposible convertir a binario'
    if (numero == 0):
            return '0'
```

```
    numeroBinario = ''
    while numero > 1:
            numeroBinario = str(numero % 2) + numeroBinario
            numero /= 2

    return '1' + numeroBinario
```
#La conversión se realiza dividiendo el número por la base (2). Los restos obtenidos forman el número binario hasta el último cociente. Por ejemplo, del número 18:

18	2	9	2	4	2	2	2
	9		4		2		**1**
0		**1**		**0**		**0**	

```
class HolaWxPython4(wx.Frame):
    def __init__(self, parent, identificador, titulo):
        wx.Frame.__init__(self, parent, identificador, titulo, size=(400, 350))
        numero = 18
        barraMenu = wx.MenuBar()
        fichero = wx.Menu()
        salir = wx.MenuItem(fichero, 1, '&Salir\tCtrl+Q')
```
#En este ejemplo hemos añadido al menú **Fichero** el submenú **Cambiar número**.
```
        cambiar = wx.MenuItem(fichero, 3, '&Cambiar numero\tCtrl+C')
        salir.SetBitmap(wx.Bitmap('imagenes/exit.png'))
        fichero.AppendItem(cambiar)
        fichero.AppendItem(salir)
        barraMenu.Append(fichero, '&Fichero')

        ayuda = wx.Menu()
        acerca = wx.MenuItem(fichero, 2, 'Acerca de')
        acerca.SetBitmap(wx.Bitmap('imagenes/acerca.png'))
        ayuda.AppendItem(acerca)
        barraMenu.Append(ayuda, 'A&yuda')
        self.SetMenuBar(barraMenu)

        self.Bind(wx.EVT_MENU, self.OnSalir, id=1)
        self.Bind(wx.EVT_MENU, self.OnAcerca, id=2)
```
#Debemos asignar al nuevo submenú un manejador que realice el procesamiento oportuno cuando el usuario haga clic sobre él.
```
        self.Bind(wx.EVT_MENU, self.OnCambiar, id=3)
```
#Creamos un panel y ubicaremos los widgets siguiendo la distribución de una caja horizontal.
```
        miPanel = wx.Panel(self, -1)
        miCajaHorizontal = wx.BoxSizer(wx.HORIZONTAL)
```
#Creamos una fuente Comics Sans de tamaño 14.

```
        miFuente = font1 = wx.Font(14, wx.SWISS, wx.NORMAL, wx.NORMAL, False, u'Comic Sans MS')
```

#El Widget texto estático wx.StaticText tiene como ventana padre el panel (nunca puede ser la caja que no es una ventana) y como identificador el 4. El tercer argumento será el texto que queremos que presente por pantalla y es una colección de información respecto al número: representación binaria, hexadecimal, divisores, etc.

```
        miTextoEstatico1 = wx.StaticText(miPanel, 4, 'Informacion sobre el numero 18.' + representacion(numero)+ '\nDivisores: ' + divisor(numero) + '\n' + primo(numero) + factorial(numero), style=wx.ALIGN_LEFT)
```

#Asignamos como fuente del widget texto estático, la que creamos previamente.

```
        miTextoEstatico1.SetFont(miFuente)
```

#Añadimos a la caja el texto estático e indicamos que el control utilice todo el espacio disponible y que se posicione a 29 unidades del lado superior e inferior.

```
        miCajaHorizontal.Add(miTextoEstatico1, -1, wx.EXPAND | wx.TOP | wx.BOTTOM, 29)
        miPanel.SetSizer(miCajaHorizontal)

        self.Centre()
        self.Show(True)


    def OnAcerca(self, event):
        miInfo = wx.AboutDialogInfo()
        miInfo.SetIcon(wx.Icon('imagenes/acerca.png', wx.BITMAP_TYPE_PNG))
        miInfo.SetName('Hola WxPython')
        miInfo.SetVersion('0.1')
        miInfo.SetDescription('Mostrando WxWidgets')
        miInfo.AddDeveloper('Joe')
        miInfo.AddTranslator('John Smith')
        miInfo.AddDocWriter('Marta Nunez')
        miInfo.SetLicence('GPLv3')
        miInfo.SetWebSite('http://quickandhandytips.com')
        wx.AboutBox(miInfo)

    def OnSalir(self, event):
        self.Close()
```

#Definimos el manejador **OnCambiar** que responderá al evento hacer clic en el submenú cambiar.

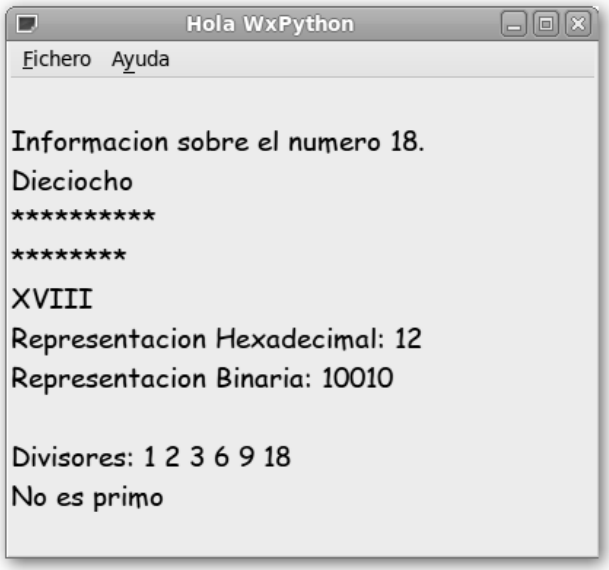
```
    def OnCambiar(self, event):
        dialogo = wx.TextEntryDialog (None, 'Elige numero', 'Por favor, introduce un numero')
```

```
        dialogo.ShowModal()
#Convertimos el texto introducido por el usuario a un valor numérico.
        numero = convierte(dialogo.GetValue())

#Accedemos a miTextoEstatico1 (id=4) para escribir sobre él.
        self.FindWindowById(4).SetLabel('Informacion sobre el numero ' + dialogo.GetValue() + representacion(numero) + '\nDivisores: ' + divisor(numero) + '\n' + primo(numero) + factorial(numero))
        self.Refresh()
        dialogo.Destroy()

app = wx.App()
HolaWxPython4(None, -1, 'Hola WxPython')
app.MainLoop()
```

En la figura adjunta se muestra el resultado de la ejecución:



26.6. Módulos para todas las necesidades.

Si en algo destaca Python es por la gran cantidad de módulos que tenemos a nuestra disposición, entre otros:

- GUI: Tkinter; wxWidget ***http://wxpython.org/***; y pyGtk, ***http://www.pygtk.org/***.
- Científicas: NumPy, ***http://numpy.scipy.org/***; DISLIN, ***http://www.mps.mpg.de/dislin/***; matplotlib, ***http://matplotlib.sourceforge.net/*** y SciPy, ***http://www.scipy.org/***.

- Bases de datos: MySQLdb, ***http://sourceforge.net/projects/mysql-python*** y PyGreSQL, ***http://www.pygresql.org/***.

- Otros: Procesamiento de imágenes: **Python Imaging Library**, ***http://www.pythonware.com/products/pil/***; Acceso a GMail, **libgmail**, ***http://libgmail.sourceforge.net/***; API de Google, **Google Web API**: **PyGoogle**, *http://pygoogle.sourceforge.net/*; Procesamiento del lenguaje natural, **NLTK**, ***http://www.nltk.org/***; desarrollo de juegos con **Pygame**, ***http://www.pygame.org/news.html***, etc.

Veamos un pequeño ejemplo de uso de módulos:

```
import math, calendar, datetime, os, webbrowser

#Para calcular el área de un círculo precisamos hacer uso de librería
math. El área de un círculo de radio r, es Pi * r2. La potencia se escribe
como (r**2), sin embargo si la eficiencia es importante en tu proyecto,
considera realizar (radio*radio).
def area(radio):
  return math.pi * (radio**2)

#Desde Python podemos ejecutar órdenes en la SHELL de Linux
hacienda uso de la librería os. En este caso mostramos a Tux
mostrando una galleta de fortuna. El resultado es un fichero abierto
conectado al descriptor de fichero miFichero que leemos línea a línea.
def verCookies():
  miFichero=os.popen("fortune | cowsay -f tux")
  for linea in miFichero.readlines():
    print linea

#Otro ejemplo de utilización de librerías es la siguiente función
consultar la Wikipedia. Lanzará el navegador por defecto y navegará a
la entrada de la Wikipedia relativa a la palabra que toma como
argumento.
def consultarWikipedia(palabra):
  url = 'http://es.wikipedia.org/wiki/' + palabra
  webbrowser.open(url)


if __name__ == '__main__':
#Mostramos el calendario del año 2009
  print calendar.calendar(2009)
  print area(5)
#Muestra la fecha actual.
  print datetime.date.today()
  verCookies()
  consultarWikipedia('Linux')
```