

Documentación Externa

TP2

TI-3404 Lenguajes de Programación.

Prolog en Python.



Brandon Carter	201114390
Andrés Maroto	201128337
David Murillo	201108648
Adrián Siles	201136841
Ariel Arias.	201136814

Profesor: Andrei Fuentes L.

15 Octubre 2013.

Tabla de Contenidos

• Descripción del problema.....	3
• Diseño del programa	4
• Librerías usadas.....	8
• Análisis de resultados.....	9
• Manual de usuario.....	10
• Conclusión personal.....	20
• Bibliografía Consultada.....	21

Descripción del Problema

El problema principal que se presenta en esta tarea programada es el de implementar el lenguaje de programación prolog utilizando otro lenguaje de alto nivel, en nuestro caso Python.

A partir de éste se encuentran dos subproblemas. El primero es el analizador sintáctico y léxico que va a contener el lenguaje. El segundo, el proceso de unificación de los predicados de la consulta con los de la base de conocimientos.

Estos son los subproblemas que se relacionan con los problemas principales.

Que el usuario agregue hechos o reglas a la base de conocimientos utilizando una sintaxis sencilla de prolog. Esos hechos o reglas pasarán por un analizador léxico(scanner) y después por uno sintáctico(parser). Si el análisis resulta correcto, se ingresará a la base de conocimientos. En caso contrario le indicará al usuario que hay algún error de sintaxis o léxico.

Que el usuario logre realizar consultas similares a las que se hacen en el sistema interactivo de Prolog, una vez que su sintaxis sea correcta se revisará en la base de conocimientos los predicados que se logren unificar con los de la consulta. Revisa primero el funtor, la aridad y por último unifica .

Diseño del Programa

Diseñamos el programa dividiéndolo en 2 grandes áreas de trabajo independientes:

1. Modo Administración
2. Modo Consulta.

Vamos primero a explicar cómo desarrollamos el **modo administrativo** que es la primera parte que implementamos.

Cuando en el menú principal decidimos que queremos ir al modo administrativo se nos abren 4 posibilidades adonde podemos ir:

- Ingresar un nuevo hecho.
- Ingresar una nueva regla.
- Ver Base de conocimientos.
- Volver al menú principal.

Primero que todo, antes de empezar a explicar cómo se guardan hechos o reglas teníamos que decidir cómo íbamos a guardar en memoria la base. Decidimos usar el tipo de datos de lista para almacenar la base de conocimientos de manera temporal, si se cierra el programa la lista se borra. Cada vez que lo abrimos debemos ingresar hechos o reglas para llenar la lista. Recordemos que siempre la BC tendrá por defecto las built-in functions write, nl y fail. La lista va a ser global por lo que su alcance es en todo el programa.

```
global BaseConocimientos
```

```
BaseConocimientos=['nl','fail','write(args)'] #Lista que contiene la BC y le agregamos los built-in predicates.
```

Ingresar un nuevo hecho: Para agregar hechos correctamente se necesita pasar por dos analizadores, el analizador léxico (scanner) y el analizador sintáctico(parser). Utilizamos una función que mandaba a llamar con el hecho que ingresábamos a estas dos funciones por separado y si las dos devuelven el valor booleano True entonces había pasado estas 2

etapas correctamente y podía ser ingresado a la lista Base con un `.append` que lo agrega al final de la lista.

```
hecho=raw_input(": ")
if ScannerHecho(hecho) and ParserHecho(hecho):
    BaseConocimientos.append(hecho)
```

La función del scanner es la más corta y lo que hace es tener en una lista los tokens que son válidos, entonces si le agregamos al hecho un token que no sea válido, osea que no esté en la lista global léxico nos devuelve un error devolviendonos el primer token inválido que encontró.

```
if caracter.lower() in Lexico[:-2]: # No usamos los últimos 2 elementos de léxico porque esos son para reglas.
    True
else:
    print("Error de scanner: "+carácter" no es token válido")
```

La función del parser analizar la posición de los tokens con un ciclo y el orden lógico que tienen para ver si es válido. Primero analizamos si el último carácter es un punto, si pasa esta prueba pasamos a verificar que el hecho empieza con minúscula con la función booleana `islower()`, si abrimos paréntesis activamos una bandera para después comprobar si los paréntesis se cerraron. El carácter después del paréntesis tiene que ser un alfanumérico o una “_”, y si tenemos una “,” el carácter después de la coma tiene que ser un alfanumérico o una “_”. Si alguna de estas pruebas fallan imprimimos a pantalla cual fue el error que se presentó específicamente. Y si pasa todas retorna `True`.

Ingresar una nueva regla: La mecánica para ingresar reglas es muy parecida a la de hechos, solo que ahora manda a llamar a tres funciones: la de parser, la del scanner y la nueva función `VerificarAridad` que devuelve `true` si es la única regla que tiene ese functor y si encuentra una que se llame igual comprueba que tengan diferente aridad. Si las 3 devuelven `true` se le da `.append` a la base de conocimientos.

La gran diferencia con respecto a los hechos se da en el Parser ya que tenemos nuevas reglas de orden sintáctico.

Lo que hicimos fue separar las diferentes partes de la regla y se las mandamos al Parser de hechos para ahorrarnos código.

Por ejemplo: `animalvivo(X):-perro(X),estavivo(X)`. Lo separamos en 3 partes: El encabezado `animalvivo(X)` y lo mandamos al parser de hechos, después verificamos que `:-` este bien y luego le hacemos un `.split` al cuerpo: `perro(X),estavivo(X)` separandolos por `,` y le mandamos al Parser de hechos por separado `perro(X)` y `estavivo(X)`.

Aquí nos enfrentamos un problema porque si la regla era esta:

`esincesto(X):-estancasados(X,Y),sonhermanos(X,Y)`. nos iba a separar con el `.split(",")` de manera errónea el cuerpo porque cada hecho tenía una `,` entre los paréntesis. Entonces lo que hicimos fue hacer una función que reemplaza las `,` por `.` si se encuentra dentro de un paréntesis. De esta manera el `.split` si iba a funcionar correctamente y despues volviamos a reemplazar los `.` por `,` para meterlo de forma correcta a la base de conocimientos.

Esa es la tarea que llevan a cabo las funciones `reemplaza` y `reemplaza2`. Esta función `reemplaza` recorren el string resultante que se encuentra posterior a la posición dentro del string que corresponde a estos dos caracteres `:-`, para recorrerlo se crea un ciclo hasta alcanzar la condición de parada que es el largo del string. Dentro del ciclo se tienen dos apuntadores que se asignan con las posiciones de los correspondientes `(` y `)`, de forma que, cuando se encuentran las posiciones de los paréntesis se realice el `replce` de `,` por `.` y asi obtener el string como se desea (esto se hace mediante el uso de slicing de listas), luego se continúa analizando a partir de la siguiente posición luego del paréntesis de cierre. Se corre el ciclo hasta alcanzar la condición de parada. La función `reemplaza2` debe retornar el string al estado anterior, como se deben regresar a su normalidad los `.` por `,`; se implementa un algoritmo en `reemplaza2` que básicamente, recorre la lista que se genera al aplicar `.split` al resultado de `reemplaza`, se recorre con el fin de ir elemento por elemento cambiando `.` por `,`. Debido a la inmutabilidad de los strings, se recorre la lista, se hace el `replce` (`.`, `,`) y se inserta en la nueva lista de retorno; que contiene la forma correcta de los strings que se manejan.

Mostrar base de conocimientos: Función sencilla que mediante un ciclo

recorre cada elemento de la lista y lo va imprimiendo a consola para poder ver qué elementos tenemos en la BC.

Ahora vamos a proceder a explicar la segunda parte de la tarea, **el modo de consulta**.

Una vez que se ingresa al modo de consulta, se muestra ‘?-’ para que el usuario ingrese la consulta, la que luego pasa por el parser y el scanner para verificar que el léxico utilizado y la estructura de la misma cumpla con el patrón de prolog, luego la consulta se transforma a una lista para ser comparada con los elementos de la base de conocimientos los cuales también se transforman en listas para ser comparadas mediante un ciclo y así verificar su unificación. Esto sucede mediante un algoritmo que verifica tanto el functor como la aridad de la consulta y de los elementos de la base el cual retorna un True si unifican todos los argumentos tomando en cuenta todas las reglas de unificación de prolog tanto para variables como constantes, las cuales son verificadas por una función la cual me retorna si el argumento es variable o constante, una vez que esta unifica o no, el motor de inferencia me retorna “YES” si el hecho existe y unifica, o un “NO” para el caso contrario. Lo anterior es para la consulta de un hecho, en el caso de las reglas se verifica primero el functor de la regla la cual es identificada por su “:-”, una vez que esta unifica con una de la base de conocimientos se realiza el mismo procedimiento explicado anteriormente para cada hecho que compone la regla, retorna también un “YES” si se cumple la regla o un “NO” en el caso contrario. En el caso de las built-in functions las cuales son *fail*, *nl* y *write(args)* también son verificadas en el momento que se van verificando los hechos de las reglas, en el caso que alguno sea una built-in function se llama a una función la cual me retorna lo que se debe hacer para cada built-in function según sea el caso, por ejemplo: para el *nl* me imprime una nueva línea, el *fail* termina la consulta y el *write* muestra en consola los argumentos que reciba para escribir.

Librerías Usadas

Para esta tarea programada no se utilizó librerías externas de python, pero si se utilizaron módulos internos del lenguaje, especialmente métodos para trabajar los strings.

- `.split()`: Retorna una lista de todas las palabras de un string usando como separador un símbolo de la misma, convirtiendo las palabras en elementos de la lista.
- `.lower()`: Convierte todos los caracteres de una cadena de texto a minúsculas.
- `.replace()`: Reemplaza una serie de caracteres por otros.
- `.isalnum()`: Retorna True si su argumento contiene números o letras del alfabeto.
- `.partition()`: Retorna una tupla de 3 elementos; la parte antes del separador, el separador mismo, y lo que está después del separador.

Análisis de Resultados

Se logran resolver con éxito los siguientes problemas de la tarea:

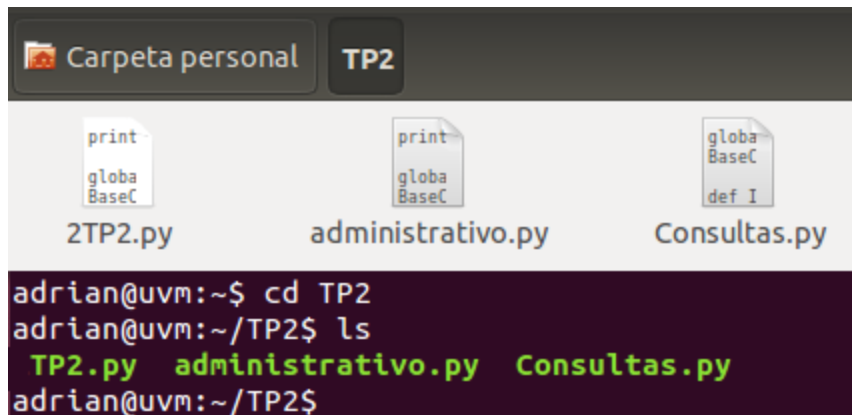
- Que el usuario ingrese un hecho o una regla a la base de conocimientos con la sintaxis adecuada.
- Unificar de manera correcta los hechos y las reglas.
- Unificar constantes con variables unbound.
- Unificar variables con variables.
- Verificar que las constantes de los predicados de la consulta sean las mismas que las de los predicados de la base de conocimientos.

Manual de Usuario

En el siguiente se mostrará los pasos básicos que debe seguir el usuario para abrir y utilizar el programa. Para iniciar el programa, se comienza por abrir la terminal del sistema operativo. Lo primero que se debe hacer, es localizar el archivo donde esta ubicado el programa, para ello utilizamos el comando “ls” y este nos dará la lista de los archivos y directorios que contiene el directorio actual:

```
adrian@uvm:~$ ls
apt-get  Documentos  ejemplo.sml~  examples.desktop  Música  Público  Ubuntu One
Descargas ejemplo.sml  Escritorio    Imágenes          Plantillas TP2  Vídeos
adrian@uvm:~$
```

Con el comando “cd”, seguido del nombre del directorio donde se encuentra ubicado el programa, nos llevará a ese directorio. Volvemos a utilizar el comando “ls” para obtener el nuevo menú del directorio en el que nos encontramos. En este caso, el programa que se debe de encontrar es TP2.py y este se encuentra en la carpeta “TP2”.



The image shows a file manager window with a dark theme. The top bar shows 'Carpeta personal' and 'TP2'. Below, three files are visible: '2TP2.py', 'administrativo.py', and 'Consultas.py'. Each file icon has a 'print' button and a 'global BaseC' label. Below the file manager is a terminal window with the following commands and output:

```
adrian@uvm:~$ cd TP2
adrian@uvm:~/TP2$ ls
TP2.py  administrativo.py  Consultas.py
adrian@uvm:~/TP2$
```

Para ejecutar el programa, se debe de realizar mediante el uso del siguiente comando “python ”, seguido del nombre de TP2.py con esto

iniciará el programa.

```
adrian@uvm:~/TP2$ ls
administrativo.py Consultas.py TP2.py
adrian@uvm:~/TP2$ python TP2.py
Lenguaje de Programacion PROLOG
Menu de opciones:
1) Modo Administrativo
2) Modo Consulta

Cual opcion desea hacer? :
```

Al iniciar el programa, por defecto el menú de acciones será mostrado, dando las opciones de: Modo Administrativo y Modo Consulta. Cada opción posee un número a la izquierda el cual lo identifica y este se utilizará para escoger la opción deseada.

```
Lenguaje de Programacion PROLOG
Menu de opciones:
1) Modo Administrativo
2) Modo Consulta

Cual opcion desea hacer? : 
```

Para indicar el modo que se desea acceder se debe indicar con el número correspondiente. Para Modo Administrativo '1' y para Modo Consulta '2'.

```
Cual opcion desea hacer? : 1

Modo Administrativo
1) Ingresar un nuevo hecho
2) Ingresar una nueva regla
3) Mostrar Base de Conocimientos
4) Regresar a Menu Principal
Que desea hacer? : 
```

1- Modo Administrativo: En este módulo se desplegarán cuatro diferentes opciones: Ingresar un nuevo hecho, Ingresar una nueva regla, Mostrar

Base de Conocimientos y Regresar al Menú Principal. De la misma forma que en el menú anterior elegimos la opción deseada con su número identificador respectivo.

A) Ingresar un nuevo hecho: Cuando se elige la opción, este indicara las reglas básicas que debe cumplir un hecho, las cuales consisten en terminar con un punto “.” y que puede tener 0 o más argumentos. Luego queda a la espera del hecho que se desea ingresar, denotado por “:” , hasta que sea ingresado o el programa se detenga .

```
Modo Administrativo
1) Ingresar un nuevo hecho
2) Ingresar una nueva regla
3) Mostrar Base de Conocimientos
4) Regresar a Menu Principal
Que desea hacer? : 1

Ingresar Hechos
*Un hecho debe terminar con un punto(.) al final
*Un hecho puede tener 0 a N argumentos
Ejemplo: perro(bruno). amigo(daniel,felipe).

: █
```

Cuando se ingresa un hecho que cumpla los lineamientos, como por ejemplos “perro(ruffo).” o “gato(motita)”, los cuales son correctos, por lo tanto el programa nos indica que ha sido ingresado a la Base de Conocimientos y el programa mostrará la opción para ingresar un nuevo hecho o para regresar al menú administrativo.

```
Ingresar Hechos
*Un hecho debe terminar con un punto(.) al final
*Un hecho puede tener 0 a N argumentos
Ejemplo: perro(bruno). amigo(daniel,felipe).

: perro(ruffo).
Hecho ingresado
hecho ingresado, presione 1 para agregar nuevo hecho y 2 para volver al menu:
```

Si se desea ingresar un nuevo hecho, ingresamos 1 y seguidamente el

programa volverá a imprimir ":" en señal de que esta esperando el ingreso del nuevo hecho. de la misma forma que antes, presentará las opciones de ingresar uno nuevo o regresar al menú administrativo.

```
Ingresa Hechos
*Un hecho debe terminar con un punto(.) al final
*Un hecho puede tener 0 a N argumentos
Ejemplo: perro(bruno). amigo(daniel,felipe).

: perro(ruffo).
Hecho ingresado
hecho ingresado, presione 1 para agregar nuevo hecho y 2 para volver al menu: 1
: gato(motita).
Hecho ingresado
hecho ingresado, presione 1 para agregar nuevo hecho y 2 para volver al menu:
```

Si se desea regresar al menú administrativo, ingresamos 2 y nos llevará hasta el menú administrativo de nuevo.

```
: perro(ruffo).
Hecho ingresado
hecho ingresado, presione 1 para agregar nuevo hecho y 2 para volver al menu: 1
: gato(motita).
Hecho ingresado
hecho ingresado, presione 1 para agregar nuevo hecho y 2 para volver al menu: 2

Modo Administrativo ←
1) Ingresar un nuevo hecho
2) Ingresar una nueva regla
3) Mostrar Base de Conocimientos
4) Regresar a Menu Principal
Que desea hacer? :
```

B) Ingresar una nueva regla: Cuando se elige la opción, este indicara algunas reglas básicas que debe cumplir una regla, las cuales consisten en terminar con un punto “.”, que puede tener 0 o más argumentos y que debe tener al menos un antecedente. Luego queda a la espera de la regla que se desea ingresar, denotado por “:”, hasta que sea ingresado.

```
Modo Administrativo
1) Ingresar un nuevo hecho
2) Ingresar una nueva regla
3) Mostrar Base de Conocimientos
4) Regresar a Menu Principal
Que desea hacer? : 2

Ingresar Reglas
*Una Regla debe terminar con un punto(.) al final
*Una Regla puede tener 0 a N argumentos
*Toda Regla puede tener N antecedentes
Ejemplo: animal(X):-perro(X),estavivo(X).

: 
```

Cuando se ingresa una regla que cumpla los lineamientos, como por ejemplo “animal(X):-gato(X),estavivo(X).”, la cual esta correcta, por lo tanto el programa nos indica que ha sido ingresado a la Base de Conocimientos y el programa mostrará la opción para ingresar una nueva regla o para regresar al menú administrativo.

```
Ingresar Reglas
*Una Regla debe terminar con un punto(.) al final
*Una Regla puede tener 0 a N argumentos
*Toda Regla puede tener N antecedentes
Ejemplo: animal(X):-perro(X),estavivo(X).

: animal(X):-gato(X),estavivo(X).
Regla ingresada
Regla ingresada, presione 1 para agregar nueva regla y 2 para volver al menu: 
```

La opción de ingresar una nueva regla y de volver al menú administrativo, funciona de la misma forma que para la opción de “Ingresar hechos”. Ingresando 1 para ingresar una nueva regla o 2 para regresar al menú administrativo.

```
: animal(X):-gato(X),estavivo(X).
Regla ingresada
Regla ingresada, presione 1 para agregar nueva regla y 2 para volver al menu: 1
: amigo(X,Y):-estavivo(X),estavivo(Y).
Regla ingresada
Regla ingresada, presione 1 para agregar nueva regla y 2 para volver al menu: 
```

```
: amigo(X,Y):-estavivo(X),estavivo(Y).
Regla ingresada
Regla ingresada, presione 1 para agregar nueva regla y 2 para volver al menu: 2

Modo Administrativo  ←
1) Ingresar un nuevo hecho
2) Ingresar una nueva regla
3) Mostrar Base de Conocimientos
4) Regresar a Menu Principal
Que desea hacer? : 
```

C) Mostrar Base de Conocimientos: Cuando se elige la opción, este mostrará todas las reglas y todos hechos que se han ingresado a la BC de forma correcta. Se mostrarán el en mismo orden cronológico que fueron ingresados.

```
Modo Administrativo
1) Ingresar un nuevo hecho
2) Ingresar una nueva regla
3) Mostrar Base de Conocimientos
4) Regresar a Menu Principal
Que desea hacer? : 3

Base de Conocimientos ←

nl
fail
write(args)
perro(ruffo).
gato(motita).
animal(X):-gato(X),estavivo(X).
amigo(X,Y):-estavivo(X),estavivo(Y).
```

D) Regresar al Menú Principal: Cuando se elige la opción, el programa regresará al Menú Principal.


```
Modo Administrativo
1) Ingresar un nuevo hecho
2) Ingresar una nueva regla
3) Mostrar Base de Conocimientos
4) Regresar a Menu Principal
Que desea hacer? : 4
Menu de opciones:
1) Modo Administrativo ←
2) Modo Consulta

Cual opcion desea hacer? :
```


2- Modo Consulta: En este módulo desplegará el simbolo “?-” que se muestra cuando el modo consulta esta a la espera de una consulta que ingrese el usuario, o bien la palabra reservada “menu” para regresar al menú principal.


```
Menu de opciones:
1) Modo Administrativo
2) Modo Consulta

Cual opcion desea hacer? : 2

ingrese: menu para regresar al menu principal
?- 
```


Una vez que se han ingresado hechos y reglas a la base de conocimientos; se podrán realizar consultas sobre las mismas. Esto mediante el uso del Modo consulta.


```
Modo Administrativo
1) Ingresar un nuevo hecho
2) Ingresar una nueva regla
3) Mostrar Base de Conocimientos
4) Regresar a Menu Principal
Que desea hacer? : 3


Base de Conocimientos 

nl
fail
write(args)
perro(bruno).
perro(ruffo).
perro(chuleta).
perro(X):-animal(X),estavivo(X).
estavivo(ruffo).
estavivo(chuleta).
animal(bruno).
animal(ruffo).
animal(chuleta).
```

Consultas:

```
Menu de opciones:  
1) Modo Administrativo  
2) Modo Consulta  
  
Cual opcion desea hacer? : 2  
  
ingrese: menu para regresar al menu principal  
?-perro(piolin)  
NO  
?- 
```

```
Menu de opciones:  
1) Modo Administrativo  
2) Modo Consulta  
  
Cual opcion desea hacer? : 2  
  
ingrese: menu para regresar al menu principal  
?-gato(ruffo)  
NO  
?-menu 
```

```
Menu de opciones:  
1) Modo Administrativo  
2) Modo Consulta  
  
Cual opcion desea hacer? : 2  
  
ingrese: menu para regresar al menu principal  
?-perro(garfield)  
NO  
?- 
```

```
Menu de opciones:
1) Modo Administrativo
2) Modo Consulta

Cual opcion desea hacer? : 2

ingrese: menu para regresar al menu principal
?-write(ruffo)      ↵
ruffo
YES
?-write(piolin)     ↵
piolin
YES
?-nl                ↵

YES
?-write(garfield)
garfield
YES
?-█
```

```
Menu de opciones:
1) Modo Administrativo
2) Modo Consulta

Cual opcion desea hacer? : 2

?-perro(X)
= ruffo;
= brun;  ↵
= theo;
?-gato(garfield)
YES
```

Conclusión Personal

Para el desarrollo de esta segunda tarea programada, decidimos trabajar con el lenguaje de programación Python, las razones fueron muchas, pero principalmente nos decidimos por este ya que teníamos buenas bases de conocimiento sobre como funcionaba ya que en el curso de Intro a la Programación utilizamos este lenguaje y era cuestión de recordar detalles para utilizarlo correctamente y sacarle provecho a la manera tan sencilla en que funciona este lenguaje interpretado.

Además como otra herramienta de trabajo resultó sumamente provechoso seguir utilizando Github que facilita el proceso a la hora de programar grupalmente. Y brinda opciones para realizar el “branching” de diferentes versiones del proyecto. Gracias a este control de versiones es que facilita la programación en un grupo porque al almacenar todas las versiones del código que se han incluido en el repositorio, resulta sencillo poder observar y comprender los cambios que realiza algún compañero y de ser necesario se pueden encontrar las versiones anteriores a un cambio. Esto sumandole que nuestro grupo de trabajo era de 5 personas y nos ayudó a organizarnos de la mejor manera.

Para desarrollar el programa en sí y entender realmente cómo funciona un lenguaje lógico para poder programarlo, nos basamos en las presentaciones que sube el profesor y en los conocimientos aprendidos en clase, por lo que no tuvimos que buscar tanto material en la web como en proyectos anteriores.

Gracias al desarrollo de esta tarea logramos ampliar y mejorar nuestros destrezas en el lenguaje de Python; adquirimos conocimientos acerca de la lógica detrás de los analizadores léxicos y sintácticos de un lenguaje y por último entendimos la mecánica de los lenguajes lógicos para resolver consultas.

También se pudo analizar y comprender las formas de resolver el problema que propone cada compañero, y con esto poder dar con la mejor combinación de ideas y obtener el mejor resultado.

Bibliografía Consultada

SWI-Prolog manual (s. f.). Recuperado el 14 de Octubre del 2013, de <http://www.swi-prolog.org/pldoc/refman/>

Python Documentation Index (s. f.). Recuperado el 14 de Octubre del 2013, de <http://www.python.org/doc/>