Instituto Tecnológico de Costa Rica
Administración de Tecnología de Información
Curso: Lenguajes de Programación

Tarea Programada 2

~Consola de Prolog en Python~

Profesor:

Andréi Fuentes Leiva.

Asistente/Tutor:

Evelyn Madriz Mata.

Elaborado por:

Christian Chaves Mora.

Sussana Ramírez Solano.

Yulay Segura Castro.

Fecha de entrega 14-10-2014

Tabla de contenido


~SWI Prolog~
File
SWI Prolog
Ir a consola
Base de Conocimientos
~Consola de Prolog en Python~


Consola de SWI Prolog en Python
08:38:08
Consola de Prolog en Python
>>
Enter

Objetivos del Proyecto

Objetivo General

Ejecutar un programa que permita mostrar una consola similar al intérprete de Prolog, que procese instrucciones y muestre resultados.

Objetivos Específicos

- Investigar funcionalidades específicas y necesarias para un intérprete.
- Confeccionar los métodos respectivos para la definición de predicados y el modo de consulta del intérprete.
- Elaborar una base de conocimientos que permita definir predicados y hechos para hacer consultas posteriormente.

Descripción del Problema.

Problema que se presenta.

El objetivo es lograr el funcionamiento de un programa el cual el usuario pueda abrir, y al hacerlo éste le muestre una consola similar al intérprete de Prolog.

El comportamiento de dicho programa debe permitir el ingreso de instrucciones, interpretarlos, procesarlos y mostrar resultados. Las instrucciones que debe soportar el lenguaje utilizado son la definición de predicados que se almacenarán en una base de conocimientos, y el modo consulta que es el análisis de dichos predicados.

Para la definición de predicados se permitirá al usuario el ingreso de hechos o reglas a la base de conocimientos, para ello es necesario utilizar la sintaxis de Prolog:

- predicado(arg n):-ant-n
- hecho (arg1, arg2)

También se debe tener en consideración varios puntos importantes que se deben respetar, tales como:

- Toda regla o hecho válido termina con un punto.
- Toda regla o hecho puede tener ***n*** argumentos.
- Toda regla puede tener ***n*** antecedentes.
- El usuario puede ingresar ***n*** cantidad de hechos o reglas a la base de conocimientos, y todas deben ser almacenadas en memoria temporal para poder ser consultadas posteriormente.

En el modo consulta el usuario interactuará por medio de consultas con el programa, y éstas consultas deben ser similares al sistema interactivo de Prolog, donde los hechos y reglas de la base de conocimientos son analizados individualmente para encontrar aquella que brinde respuesta a la consulta y así responder **YES** o **NO,** dependiendo de la información almacenada en la base de conocimientos ya especificada.

Para el modo consulta se debe mantener en consideración los siguientes puntos importantes:

- Si una regla tiene antecedentes, éstos deben ser validados para poder responder.
- Debe mantenerse un tipo de pila con los diferentes predicados.
- Se debe manejar backtracking para el proceso de resolución de metas.

Solución del Problema.

- Inicialmente se debe plantear un proceso que permita conocer los requisitos básicos deficiencias del programa, como es en el caso de la Definición de Predicados que es necesario el uso de Built-in-predicates tales como **write(arg)** y **nl,** o en el Modo Consulta que se debe manejar backtracking **.** Esto procede a una investigación de cómo usarlos, cuáles requisitos necesitarán y cuáles librerías deberán ser incluidas para su funcionamiento.
- A partir de la definición de predicados el programa debe tener una base de conocimientos que será quien permita hacer las consultas revisando los hechos y reglas que se precisaron en ella, para así poder dar respuesta a dicha consulta elaborada por el usuario.

Descripción del Programa.

- Front-end:

El programa consiste en elaborar un intérprete similar a Prolog en otro lenguaje de programación, pero que permita realizar consultas exactamente igual que en Prolog. Además que mantenga la misma sintaxis a la hora de definir reglas o predicados para ser almacenados en la base de conocimientos que es un punto clave para realizar las consultas.

Para ello el usuario deberá ser capaz de manipular el programa a su gusto y así poder definir los predicados y/o reglas que la base de conocimientos resguardará, e igualmente el usuario tendrá el control de realizar las consultas que le plazca, y para ello el programa debe brindarle una respuesta basándose en los predicados definidos con anterioridad por el mismo usuario y que se mantienen guardados en una memoria temporal de la base de conocimientos.

En otras palabras el usuario interactuará directamente con el intérprete tanto para definir los predicados como para realizar las consultas, teniendo control total sobre mismo, pero manteniendo restricciones principalmente en la sintaxis.

- Back-end:

Es la gestión, control y administración necesaria para implementar el sistema bajo los requisitos que éste demanda para su funcionamiento.

Debido que el programa consiste en la elaboración de un intérprete similar a Prolog pero utilizando otro lenguaje se deben mantener presente aspectos sumamente importantes para lograr que el cometido se cumpla a la perfección.

Para ello se debe especificar funcionalidades del programa y que el usuario debe entender al momento de utilizarlo; entre ellos están:

- La definición adecuada de predicados y/o reglas.
- La capacidad de almacenar **n** cantidad de hechos y reglas en la base de conocimientos.
- El uso de Built-in-predicates.
- Manejo de instrucciones claves, tal como </define> que se utilizará para salir del modo de definición de predicados.
- La validación de los antecedentes en el modo consulta.
- El manejo de backtracking para el proceso de resolución de metas.

Diseño del Programa.

Decisiones de Diseño:

- **Lenguaje de Programación Utilizado:**

Se decidió el uso de Python como lenguaje de programación para el proyecto debido que es un lenguaje muy "expresivo" y muy compacto, permitiendo que un programa en Python sea más corto que su equivalente en lenguajes como C, además de ser un lenguaje de programación de muy alto nivel.

Además la sintaxis de Python es muy elegante y permite la escritura de programas cuya lectura resulta fácil y puede usarse como lenguaje imperativo procedimental.
Algunas de las ventajas que visualizamos en el uso de Python para el desarrollo del programa fueron:

- **Simplificado y rápido**: Python simplifica mucho la programación ya que te propone un patrón. Además es un *gran lenguaje para scripting*.

- **Elegante y flexible:** el lenguaje brinda muchas herramientas, si se desean listas de varios datos, no hace falta declarar cada cosa, y al ser tan flexible no existe preocupación de más por los detalles.

- **Ordenado y limpio:** Python es muy leíble, cualquier otro programador lo puede leer y trabajar sobre él. Además se puede destacar que los módulos están bien organizados, a diferencia de otros lenguajes.

- **Portable:** en comparación con otros lenguajes, Python también es muy flexible con respecto en el sistema operativo que se utilice, ya que no muestra muchas restricciones ni variaciones entre un sistema operativo y otro.

Distribución del Trabajo:

Dedicamos varios días únicamente a investigación de distintas fuentes que nos permitieran desarrollar más adecuadamente el programa utilizando Python, además tuvimos complicaciones con respecto a encontrar información sustentable para el uso de Parser en el programa, sin embargo después de una ardua investigación se logró encontrar los datos necesarios para implementar las herramientas generadoras de parsers.

Entre los datos encontrados de las investigaciones realizadas se utilizó la herramienta de capas exclusivo para Python (también llamado Ply-3.4), que nos permitió elaborar el analizador léxico (.lex) y entender el analizador sintáctico (.yacc).

Para el uso de esta herramienta, primeramente fue necesario descargar la carpeta (ver Referencias Utilizadas – ply 3.4), que contenía los datos que posteriormente serían importados, seguidamente se descomprimió dicha carpeta y así se pudo visualizar algunos de los datos y códigos que ésta traía, se analizó cada código y se seleccionó como retroalimentación ciertas instrucciones que más adelante permitieron el desarrollo del analizador léxico que se utilizó en el programa de nuestro proyecto.

Para el uso del ply-3.4, además de descomprimir la carpeta, también se debe adjuntar los datos de ésta con los demás archivos .py que se utilizarían en el proyecto, debido que es una "restricción" necesaria de la herramienta para su uso y funcionamiento, principalmente por ser importada.

Algoritmos Usados:

Para la implementación del programa no creamos tantos algoritmos específicos, ya que usamos funciones de librerías del sistema que realizaban el trabajo, los algoritmos creados se detallan a continuación:

- **Def sintaxis():**

Se utilizó como medio de verificación de la sintaxis en el momento de escribir predicados y/o reglas y también durante el modo de consulta, en caso de haber un error sintáctico o léxico el programa lo indica automáticamente para que el usuario se entere y comprenda la razón por la cual no obtuvo una respuesta a su consulta o bien para que corrija la escritura del predicado que quiere almacenar en la base de conocimientos.

También se encarga de aceptar la indicación del usuario cuando desea finalizar la definición de los predicados en la base de conocimientos para iniciar con el modo consulta.

Esta función hace un llamado a la función de lexer, la cual es una modificación de los archivos .lex que estaban almacenados en la carpeta 'ply 3.4'. Esta función lexer identifica el tipo de cada variable que digita el usuario, separando cada variable en tokens y que sea más sencillo el manejo por partes y se pueda verificar cada variable por separado y si se acepta o no para entrar a la Base de Conocimientos o no.

Ejemplo:
Sólo se permiten definiciones escritas léxica y sintácticamente correctas, es decir:

```
>> perro(luna).
>> gato(motita).
```

Por lo que errores como los siguientes no serán aceptados:

```
>> perro.
>> gato(goku.
>> gato goku
```

Se adjunta el código de sintaxis()

```
def sintaxis():
 print ('Welcome to SWI-Prolog…')
 val1=input()
 datos=[]
 lexer = lex.lex()
 elementos=[]
 y=False
 boolean= False
 if (val1=='<define>'):

     while(y==False):
         print ('>>')
         val1=input()
         data=val1
         elementos=RetlexicalTYPE(data)
         paren=PARENT(data)
         contlistelem=len(elementos)-1
         if (val1=='</define>.'):
             y=True
         if (elementos[0]=='NAME'):
             if(elementos[contlistelem]=='PERIOD'):
                 if(paren==True):
                     datos.append(val1)
                 else:
                     print ('Valor invalido:Defina parametros entre paréntesis')
             else:
                 print ('Valor invalido:Recuerde agregar punto')
         else:
             print ('Valor invalido')

 if(val1=='</define>.'):
     print('?')
     val2=input()
     int=0
     x=False
     while(x==False):
         for i in range(0,len(datos)):
             cont=len(datos)-1
             while(cont>=0):
                 if(val2==datos[cont]):
                     boolean=True
                 cont=cont-1
         x=True
     if(boolean==True):
         print('Yes')
```

```
        else:
            print('No')

    else:
        print('error')
```

- **Def RetlexicalTYPE:**

Es un analizador léxico de los datos que el usuario ingrese como parte de los predicados o reglas que desea definir y almacenar en la base de conocimientos.

Es la obtención base de la herramienta ply-3.4 especificada anteriormente, y su funcionamiento es básicamente analizar que todo lo que el usuario ingreso y verificar que el léxico sea correcto y aceptable por el programa.

```
def lexicalTYPE(data):
    lexer = lex.lex()
    lexer.input(data)
    while True:
        tok = lexer.token()
        if not tok: break
        return (tok.type)

        for tok in lexer:
            return (tok.type)
def RetlexicalTYPE(data):
    lexer = lex.lex()
    lexer.input(data)
    elem=[]
    while True:
        tok = lexer.token()
        if not tok: break
        elem.append(tok.type)

        for tok in lexer:
            elem.append(tok.type)
    return (elem)
```

- **Def PARENT:**

También es parte de un analizador y su funcionamiento dentro del código es verificar que el uso de los paréntesis () sea correcto, principalmente en la definición de los argumentos correspondientes a cada hecho y regla.

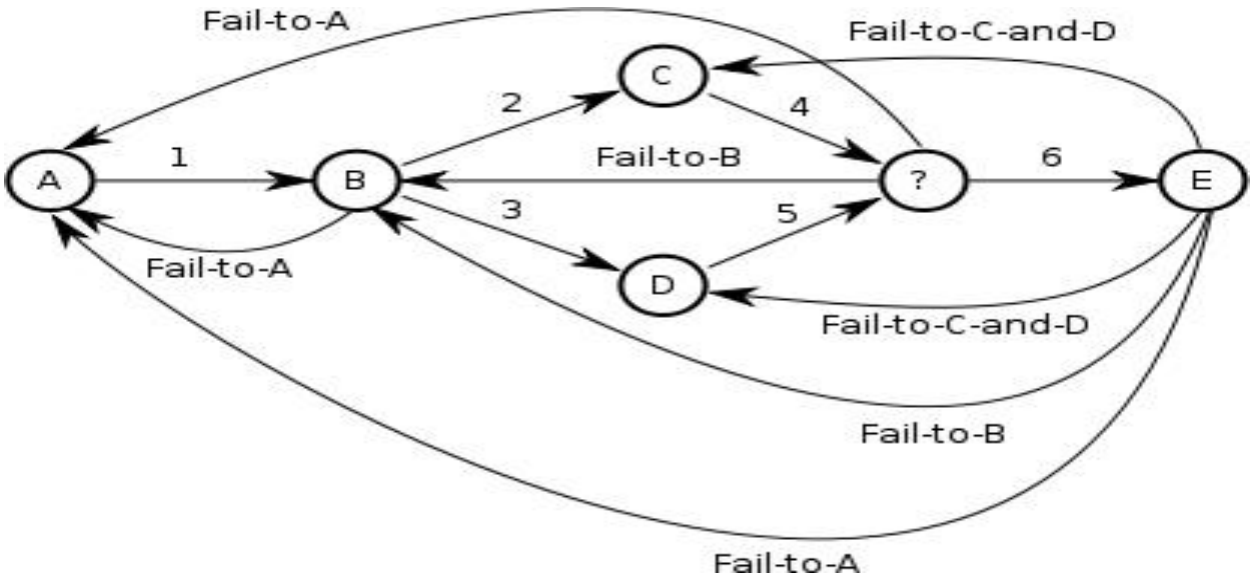
```
def PARENT(data):
    elementos=RetlexicalTYPE(data)
    cont=len(elementos)-1
    contador=len(elementos)-1
    existe=False
    while(cont>=0):
        if((elementos[cont])=='LPAREN'):
            if((elementos[contador-1])=='RPAREN'):
                existe=True
                return (existe)
        cont=cont-1
    return (existe)
```

- **Def backtracking:**

Utilizado para hacer una búsqueda sistemática a través de todas las configuraciones posibles dentro de un espacio de búsqueda, construyendo posibles soluciones candidatas de manera sistemática. Además verifica a cada elemento resguardado en la base de conocimientos con el fin de comprobar si es o no una posible solución a la consulta realizada por el usuario.

Para una mejor visualización de los algoritmos utilizados se muestra a continuación los gráficos de algunas de las funciones especificadas anteriormente:

BACKTRACKING:



Librerías y Módulos Utilizados:

- **Lib/ply.py:** más que una librería es una herramienta desarrollada en Python completamente, por lo cual es otra aplicación de la lex y yacc para Python. Algunas de las características notables incluyen el hecho de que su implementado enteramente en Python y utiliza LALR (1) de análisis sintáctico que es eficiente y muy adecuado para gramáticas más grandes.
- **Lib/glob.py:** es un módulo que busca todos los nombres de ruta coincidentes con un patrón específico de acuerdo a las reglas usadas por el Shell.
- **Lib/shutil.py:** es un módulo que ofrece una serie de operaciones de alto nivel sobre los archivos y colecciones de archivos. En particular, se proporcionan funciones que admite la copia de archivos y la eliminación de los mismos.
- **Lib/tokinize.py:** es un módulo que proporciona un escáner léxico para el código fuente de Python, implementado en Python. El escáner en este módulo devuelve los comentarios como fichas y, lo que es útil para implementar.
- **Lib/sys.py:** es un módulo que proporciona el acceso a algunas de las variables utilizadas o mantenidas por el intérprete, y que interactúan entre sí.
- **Tkinter:** es un módulo especial de Python para realizar interfaz gráfica y de esta manera era más sencillo la utilización de ~consola de Prolog~.

Referencias Utilizadas:

Durante el desarrollo del programa ~Consola de Prolog en Python~, se utilizaron las siguientes referencias bibliográficas tanto para fuente informática como para construcción de código.

elhacker.net. "Tema: [Python/Tkinter](Kyurem v2.0)Consola de comandos hecha en python". Recuperado de: http://foro.elhacker.net/scripting/pythontkinterkyurem_v20consola_de_comandos_hecha_en_python-t396560.0.html

EsCompiladores. "PLY, una Implementación de lex y yacc en Python". Recuperado de: http://escompiladores.wordpress.com/2014/03/11/ply-una-implementacion-de-lex-y-yacc-en-python/

Mclibre.org. "Introducción a la programación. Entrada y salida". Recuperado de: http://www.mclibre.org/consultar/python/lecciones/python_entrada_salida.html

Python para programadores con experiencia. "1.8 Todo sobre listas". Recuperado de: http://es.diveintopython.net/odbchelper_list.html

Python Software Foundation. "getch 1.0". Recuperado de: https://pypi.python.org/pypi/getch

- Python Software Foundation. "ply 3.4". Recuperado de: https://pypi.python.org/pypi/ply

- *Vínculo de descarga de la herramienta ply*

Análisis de Resultados:

Objetivos Alcanzados:

- Se logró crear un analizador léxico/sintáctico que verifique la sintaxis de los predicados, reglas y hechos definidos por el usuario.
- Uso correcto de las instrucciones <define> y </define> que especifican respectivamente la entrada y salida al modo de definición de predicados.
- Variedad en la cantidad de los argumentos por cada hecho o regla especificada.
- Resguardo de las reglas y hechos en la memoria temporal de la base de conocimientos.
- Funcionamiento en gran parte del modo consulta, principalmente en la validación de antecedentes de cada regla definida.
- Uso de pila para mantener distintos predicados.

Objetivos no Alcanzados:

- A pesar que tenemos archivos de pruebas con distintos códigos de backtracking, no logramos el funcionamiento adecuado del mismo.
- No implementamos el Buil-in-predicate conocido como write(arg), que se encarga mostrar en pantalla el argumento instanciado.
- Intentamos realizar el proyecto utilizando el lenguaje C++, sin embargo no logramos obtener una buena adaptación con el mismo.

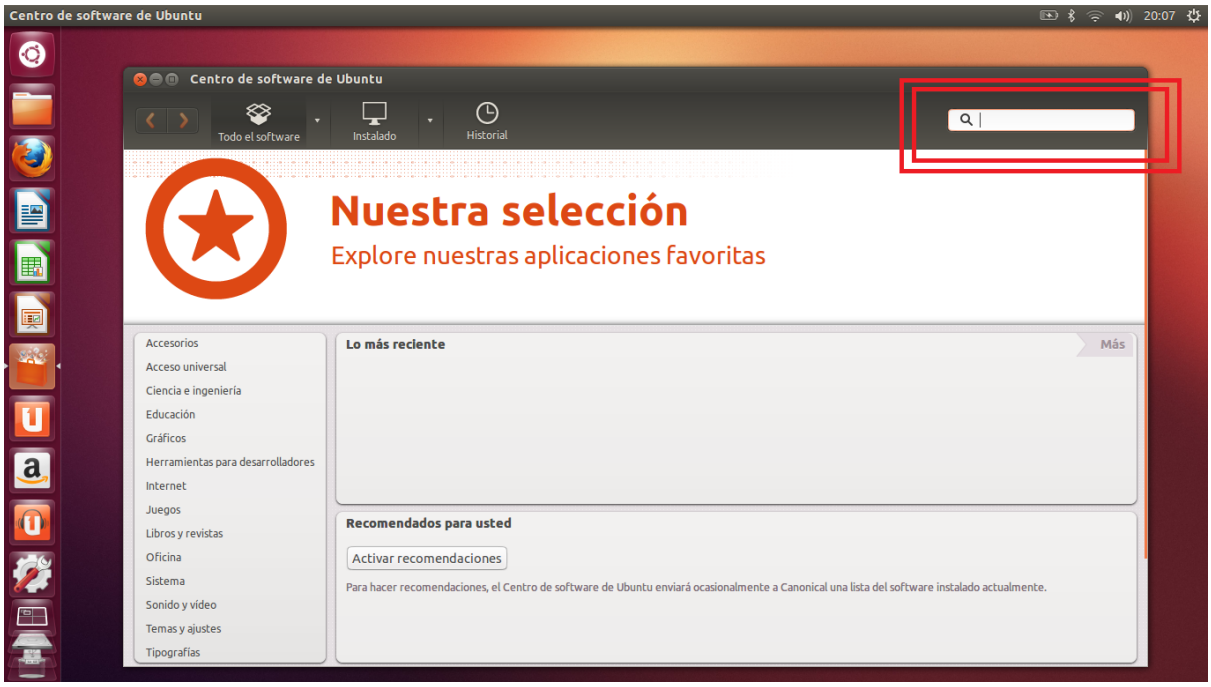
Propuestas de solución para los objetivos no alcanzados:

- Dedicar más tiempo investigativo para conocer más a fondo el lenguaje C++.
- Distribuir de mejor manera el tiempo, y tener una mejor organización grupal con respecto a las responsabilidades de cada quien con el proyecto.
- Más empeño y proactividad.

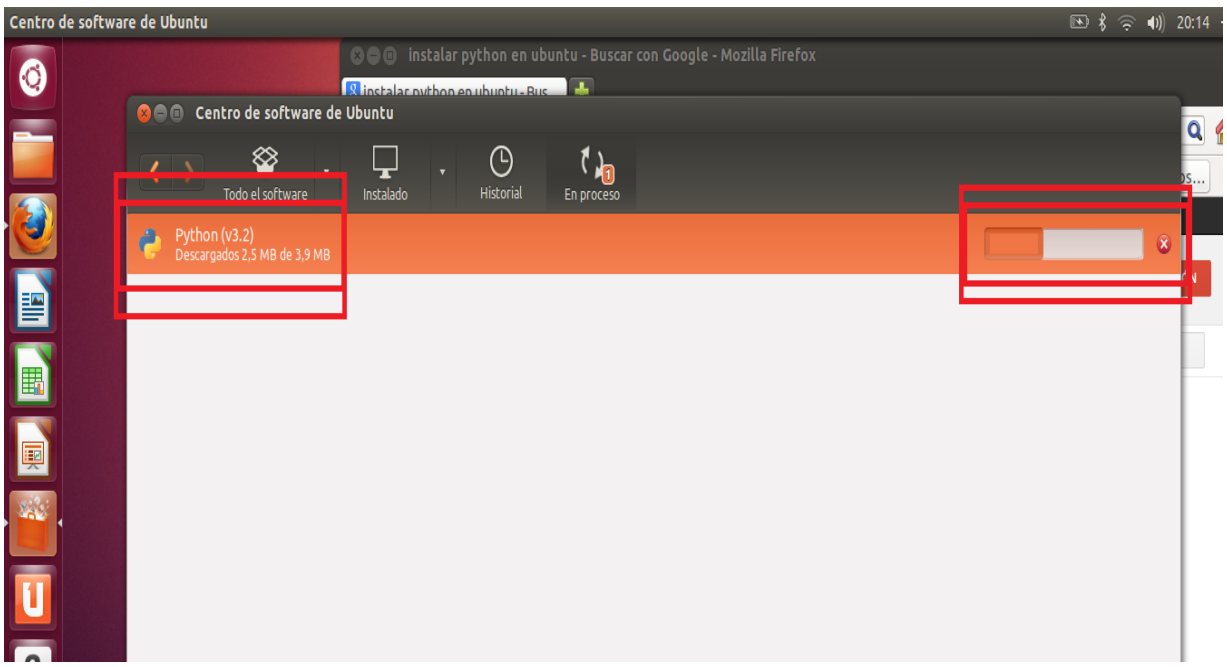
Manual de Usuario:

1. Iniciamos sesión en nuestro sistema operativo de Linux, en este caso Ubuntu.
2. En la pantalla inicial, nos vamos al Panel, a la pestaña Centro de Software de Ubuntu y lo abrimos.

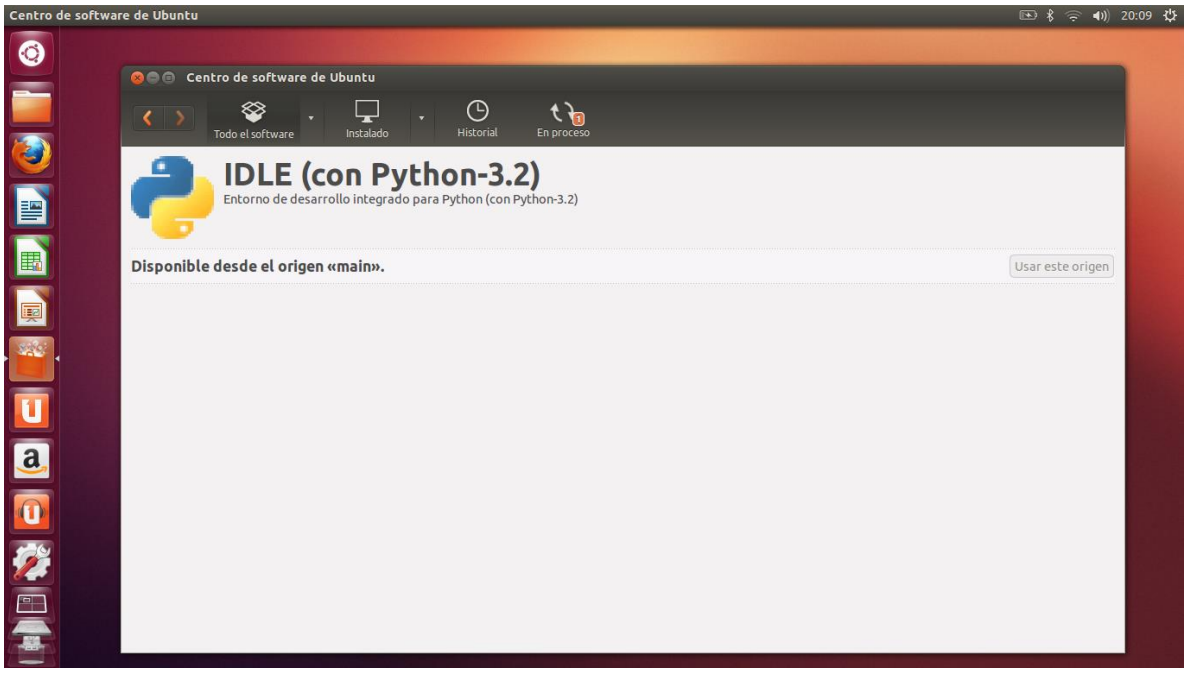
3. Una vez abierta la ventana del Centro de Software, en la barra de búsqueda escribimos

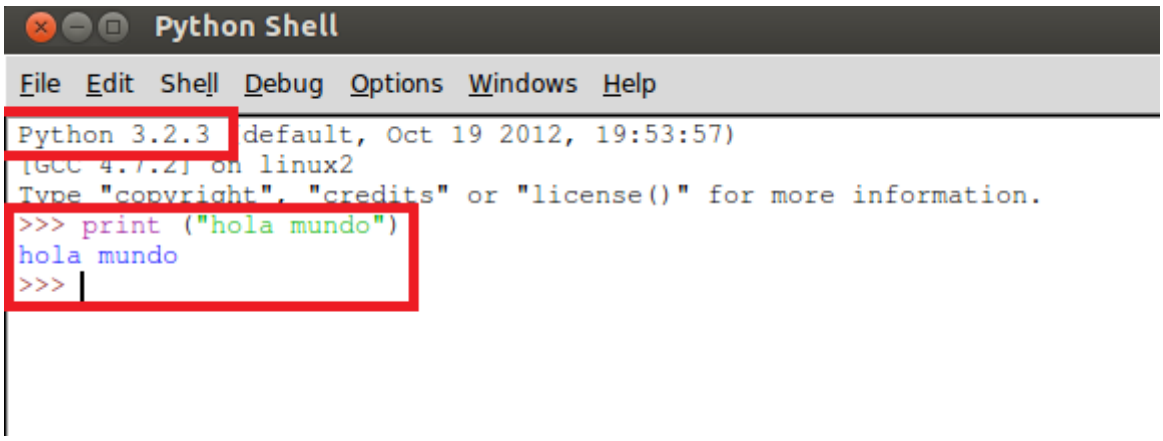


4. Instalamos el programa deseado.



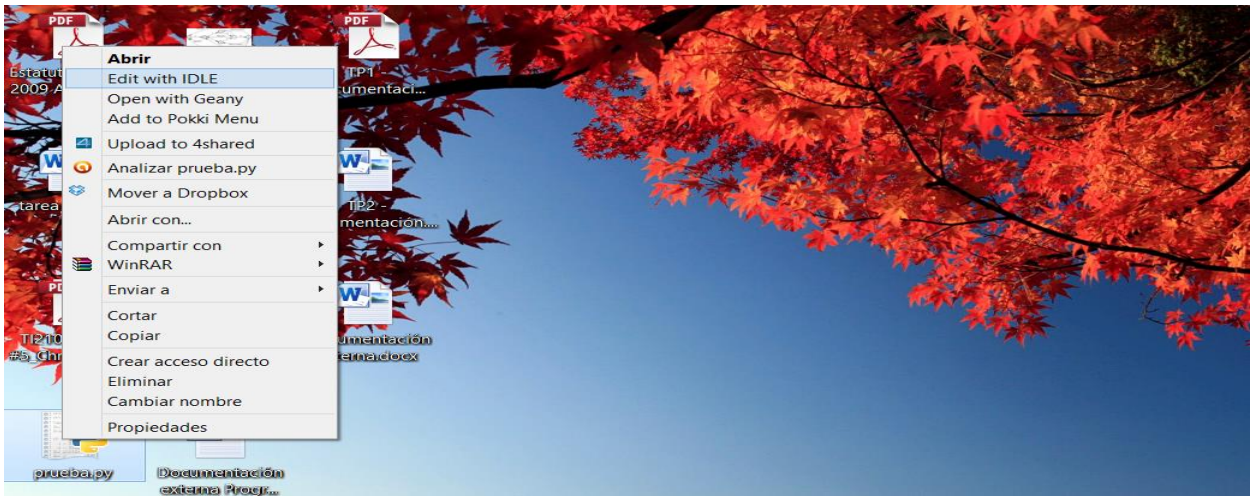
5. Ya tenemos instalado Python 3.2 y verificamos con una prueba sencilla, escribimos:
   print(“hola mundo”)



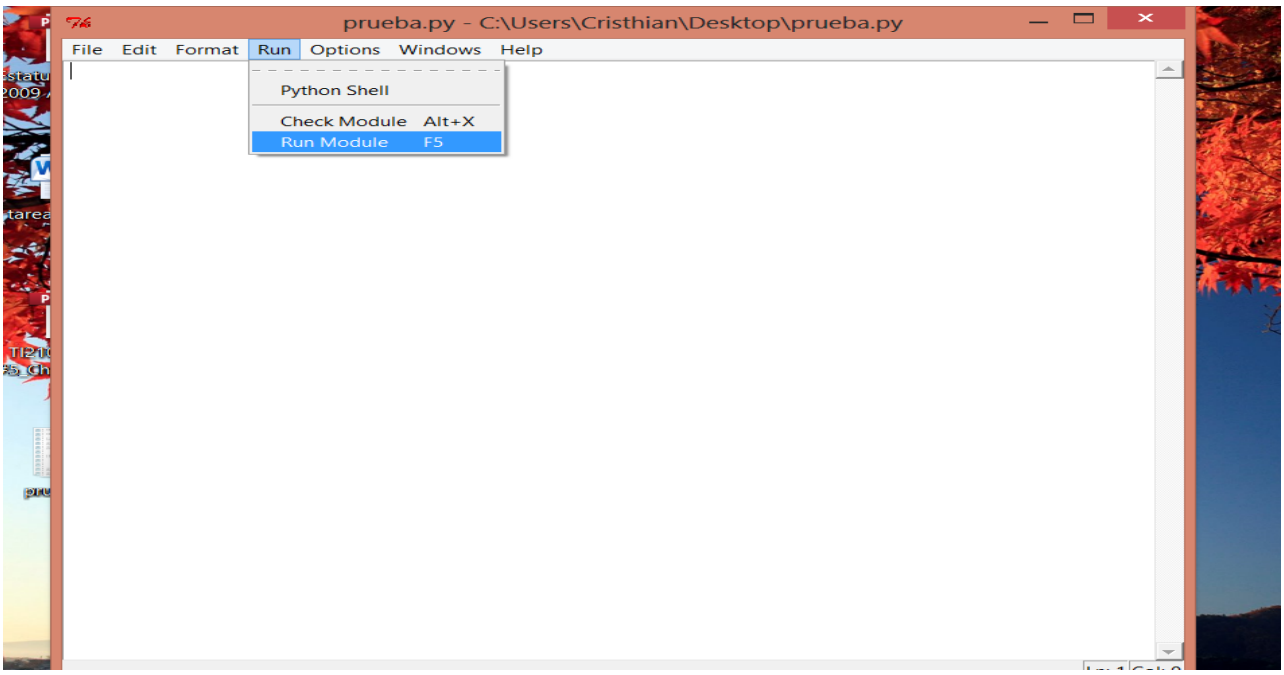


6. Instala los paquetes que acabas de descargar para agregar automáticamente el soporte de Python en su computadora.

7. Para el uso del intérprete, inicie Python presionando click derecho sobre el programa y seguidamente seleccione la opción Edit with IDLE.

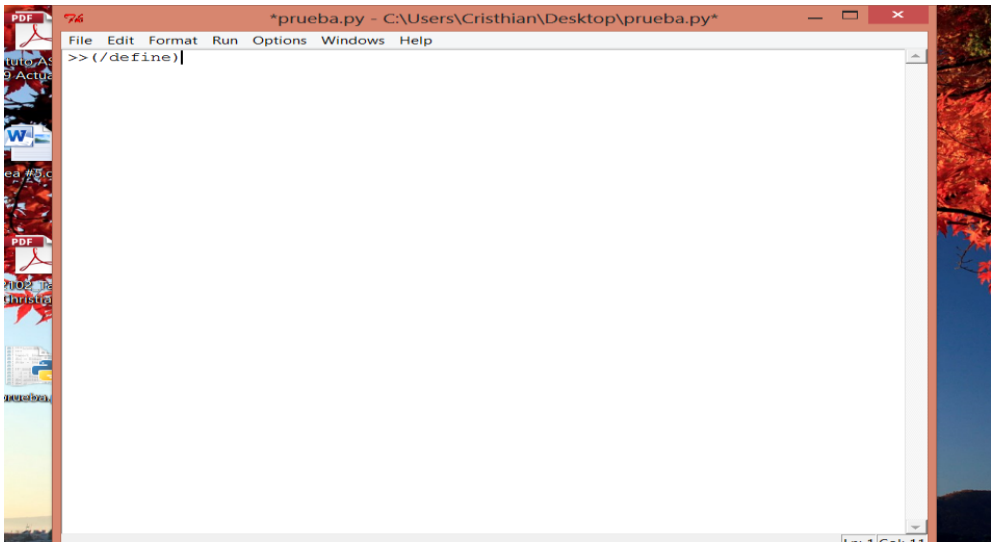


8. Cuando el IDLE de Python esté abierto diríjase a la viñeta Run, presione click sobre ella y seleccione la opción Run Module, o bien solo presione la tecla F5.



9. Aparecerá un cuadro donde podrá escoger la opción para definir predicados y otra opción para el modo consulta, seleccione la opción de Definir predicados y seguidamente escriba las reglas o hechos que desee respetando las leyes sintácticas y léxicas del mismo.

10. Para salir del modo de definición de predicados escriba la instrucción </define> y con eso el sistema saldrá del modo de definición de predicados.



11. Para el uso de modo consulta, seleccione la opción Modo Consulta que aparece en el cuadro, y prosiga a realizar las consultas que desee. Para finalizar el uso del programa solo presione la equis (X) roja en la esquina superior derecha.

Conclusión Personal:

El objetivo de la tarea programada es bastante interesante ya que nos enseñó cómo elaborar un intérprete de un lenguaje de programación con el cual podremos interactuar para realizar distintas consultas con respecto a una base de conocimientos anteriormente definida.

Nos sentimos realmente satisfechos con el proyecto ya que nos mostró un mundo más allá que simplemente escribir código, sino que nos dio el “poder” para controlar la escritura de dicho código por parte de otros usuarios.
Además durante la realización de la tarea programada desarrollamos mayormente nuestro intelecto investigativo al tener que indagar más profundamente sobre ciertos criterios de Python que desconocíamos.

Fue realmente gratificante observar los resultados de los esfuerzos principalmente los investigativos, porque gracias a ellos obtuvimos conocimientos e información bastante nueva y que no recordábamos de Python, principalmente con el uso de los archivos que hallamos dentro de la carpeta Ply-3.4 que fueron de gran ayuda en todos los sentidos.
Finalmente, este proyecto programado permitió mejorar las habilidades personales y descubrir debilidades tanto en la administración del tiempo, el trabajo en equipo y entre otras actividades necesarias para el desarrollo del programa, lo cual nos prepara aún más como futuros profesionales.