Diseño de Compiladores – 2015

Trabajo Obligatorio

Desarrollo de un intérprete del lenguaje Python

1. Objetivos

El objetivo de este trabajo obligatorio es la construcción de un intérprete de un subconjunto del lenguaje PYTHON. En la sección 2 se presenta una introducción al lenguaje de programación PYTHON, indicando el conjunto mínimo de aspectos que se deben incluir en el intérprete a desarrollar. En la sección 3 se presentan algunos aspectos del intérprete a construir, como ser el soporte y manejo de errores, el ambiente de ejecución, etc. Por último, en la sección 4 se presentan aspectos formales de la entrega a realizar.

2. Definición del lenguaje

En esta sección se presentan aspectos del lenguaje de programación **Python**, los cuales deberán ser tenidos en cuenta a la hora de implementar la solución del obligatorio.

Introducción

Python es un lenguaje de programación creado por Guido van Rossum a principios de los años 90 cuyo nombre está inspirado en el grupo de cómicos ingleses "Monty Python". Es un lenguaje con una sintaxis muy limpia y que favorece un código legible. Se trata de un lenguaje interpretado con tipado dinámico, fuertemente tipado, multiplataforma y orientado a objetos.

Un programa sencillo

El primer programa que vamos a escribir en Python es el clásico Hola Mundo. Este es muy sencillo, como se presenta a continuación.

```
print "Hola Mundo"
```

Los archivos con programas Python generalmente se almacenan en archivos con extensión .py. Si el intérprete que permite ejecutar estos programas, se denomina **python**, y el programa anterior se almacena en un archivo de nombre **hola.py** entonces la ejecución de este programa seria:

```
python hola.py
```

Tipos básicos

En Python los tipos básicos se dividen en:

- Números, como pueden ser: 3 (entero) o 15.57 (flotante)
- Cadenas de texto, como: "Hola Mundo"
- Valores booleanos: True (cierto) y False (falso).

Las variables pueden ser creadas implícitamente a través de una asignación. Por ejemplo:

```
# esto es una cadena
c = "Hola Mundo"

# y esto es un entero
e = 23

# podemos comprobarlo con la función type
type(c)
type(e)
```

Los comentarios en Python se escriben con #. Toda línea que comience con este carácter, es ignorada completamente por el intérprete.

Números

Los números enteros son aquellos números positivos o negativos que no tienen decimales (además del cero). En Python se pueden representar mediante el tipo **int** o el tipo **long**. La única diferencia es que el tipo **long** permite almacenar números más grandes.

Al asignar un número a una variable esta pasará a tener tipo **int**, a menos que el número sea tan grande como para requerir el uso del tipo **long**. Podemos indicar a Python que un número se almacene usando **long** añadiendo una L al final.

```
# type(entero) devolvería int
entero = 23

# type(entero) devolvería long
entero = 23L
```

Los números reales son los que tienen decimales. En Python se expresan mediante el tipo **float**. Para representar un número real en Python se escribe primero la parte entera, seguido de un punto y por último la parte decimal.

```
real = 0.2703
```

Operadores aritméticos

Se dispone de los siguientes operadores aritméticos

Operador	Descripción	Ejemplo
+	Suma	`r = 3 + 2  # r es 5`
-	Resta	`r = 4 - 7  # r es -3`
-	Negación	`r = -7  # r es -7`
*	Multiplicación	`r = 2 * 6  # r es 12`
**	Exponente	`r = 2 ** 6  # r es 64`
/	División	`r = 3.5 / 2  # r es 1.75`
//	División entera	`r = 3.5 // 2 # r es 1.0`
%	Módulo	`r = 7 % 2  # r es 1`

Es importante aclarar que Python realiza promoción de operadores. Por ejemplo, si al realizar una suma, ambos operadores son enteros, el resultado es entero. Pero si al realizar la misma operación, uno de los operandos es flotante, el resultado total de la operación será flotante.

Operadores bitwise

Se dispone de los siguientes operadores a nivel de bits. Los mismos operan solo con argumentos enteros.

Operador	Descripción	Ejemplo
&	and	`r = 3 & 2  # r es 2`
|	or	`r = 3 | 2  # r es 3`
^	xor	`r = 3 ^ 2  # r es 1`
~	not	`r = ~3  # r es -4`
<<	Desplazamiento izq.	`r = 3 << 1 # r es 6`
>>	Desplazamiento der.	`r = 3 >> 1 # r es 1`

Strings

Las cadenas de caracteres son texto encerrado entre comillas simples ('cadena') o dobles ("cadena"). Dentro de las comillas se pueden agregar caracteres especiales escapándolos con \, como \n, el carácter de nueva línea, o \t, el de tabulación.

También es posible encerrar una cadena entre triples comillas (simples o dobles). De esta forma podremos escribir el texto en varias líneas, y al imprimir la cadena, se respetarán los saltos de línea que introdujimos sin tener que recurrir al carácter \n, así como las comillas sin tener que escaparlas.

```
triple = """primera línea
                esto se verá en otra línea"""
```

Las cadenas también admiten operadores como +, que funciona realizando una concatenación de las cadenas utilizadas como operandos y *, en la que se repite la cadena tantas veces como lo indique el número utilizado como segundo operando.

```
a = "uno"
b = "dos"

c = a + b # c es "unodos"
c = a * 3 # c es "unounouno"
```

Booleanos

Una variable de tipo booleano sólo puede tener dos valores: True (cierto) y False (falso).

Operadores booleanos

Se disponen de los siguientes operadores booleanos

Operador	Descripción	Ejemplo
and	¿se cumple a y b?	`r = True and False # r es False`
or	¿se cumple a o b?	`r = True or False  # r es True`
not	No a	`r = not True  # r es False`

Asimismo, los tipos booleanos son el resultado de los operadores relacionales. Se disponen de los siguientes operadores relacionales.

Operador	Descripción	Ejemplo
==	¿son iguales a y b?	`r = 5 == 3 # r es False`
!=	¿son distintos a y b?	`r = 5 != 3 # r es True`
<	¿es a menor que b?	`r = 5 < 3  # r es False`
>	¿es a mayor que b?	`r = 5 > 3  # r es True`
<=	¿es a menor o igual que b?	`r = 5 <= 5 # r es True`
>=	¿es a mayor o igual que b?	`r = 5 >= 3 # r es True`

Tipos estructurados

En Python se soportan estructuras complejas, de las cuales nos interesan las siguientes:

- Listas
- Diccionarios
- Tuplas

Listas

La lista es un tipo de colección ordenada. Sería equivalente a lo que en otros lenguajes se conoce por arrays, o vectores. Las listas pueden contener cualquier tipo de dato: números, cadenas, booleanos, otras listas, etc.

Para crear una lista, hay que indicar entre corchetes, y separados por comas, los valores que queremos incluir en la lista:

```
l = [22, True, "una lista", [1, 2]]
```

Podemos acceder a cada uno de los elementos de la lista escribiendo el nombre de la lista e indicando el índice del elemento entre corchetes. En Python los índices de las listas comienzan en 0.

```
l = [11, False]
mi_var = l[0] # mi_var vale 11
```

Con respecto al operador [] de Python, podemos utilizarlo también con números negativos. Si se utiliza un número negativo como índice, esto se traduce en que el índice empieza a contar desde el final, hacia la izquierda; es decir, con [-1] accederíamos al último elemento de la lista, con [-2] al penúltimo, con [-3], al antepenúltimo, y así sucesivamente.

Python soporta el slicing o particionado de listas. Este mecanismo permite seleccionar porciones de la lista. Si en lugar de un número escribimos dos números inicio y fin separados por dos puntos (inicio:fin) Python interpretará que queremos una lista que vaya desde la posición inicio a la posición fin, sin incluir este último. Si escribimos tres números (inicio:fin:salto) en lugar de dos, el tercero se utiliza para determinar cada cuantas posiciones añadir un elemento a la lista.

```
l = [99, True, "una lista", [1, 2]]
mi_var = l[0:2] # mi_var vale [99, True]
mi_var = l[0:4:2] # mi_var vale [99, "una lista"]
```

Hay que mencionar así mismo que no es necesario indicar el principio y el final del slicing, sino que, si estos se omiten, se usarán por defecto las posiciones de inicio y fin de la lista, respectivamente:

```
l = [99, True, "una lista"]
mi_var = l[1:]  # mi_var vale [True, "una lista"]
mi_var = l[:2]  # mi_var vale [99, True]
mi_var = l[:]   # mi_var vale [99, True, "una lista"]
mi_var = l[::2] # mi_var vale [99, "una lista"]
```

Este mecanismo también permite modificar la lista

```
l = [99, True, "una lista", [1, 2]]
l[0:2] = [0, 1] # l vale [0, 1, "una lista", [1, 2]]
```

Diccionarios

Los diccionarios, también llamados matrices asociativas, deben su nombre a que son colecciones que relacionan una clave y un valor. Por ejemplo, veamos un diccionario de películas y directores:

```
d = {
     "Love Actually ": "Richard Curtis",
     "Kill Bill": "Tarantino",
     "Amélie": "Jean-Pierre Jeunet"
}
```

Para acceder a los elementos de un diccionario, también se usa el operador [], solo que en vez de utilizar el índice, se utiliza la clave.

```
d["Love Actually "] # devuelve "Richard Curtis"
d["Kill Bill"] = "Quentin Tarantino"
```

Tuplas

Una tupla representa un conjunto de valores asociados. Son similares a los diccionarios, pero no tienen una clave asociada. Para definirlas debemos realizar lo siguiente:

```
t = (1, 2, True, "python")
```

El constructor de las tuplas es la coma. Es importante tener en cuenta esto, ya que para crear tuplas con un solo elemento, debemos utilizar la coma para diferenciar este de un valor parentizado

```
>>> t = (1)
>>> type(t)

type "int"

>>> t = (1,)
```

```
>>> type(t)

type "tuple"
```

Para acceder a los elementos de una tupla, usamos también el operador []

```
mi_var = t[0] # mi_var es 1
mi_var = t[0:2] # mi_var es (1, 2)
```

Las tuplas son inmutables. A diferencia de listas y diccionarios, en caso de crear una tupla, la misma no puede ser alterada.

Bloques y anidamiento

En el caso de Python, el manejo de bloques y anidamiento sigue una norma poco tradicional en comparación con los lenguajes de programación tradicionales como C o Java. Lo que en Java se escribiría de esta forma:

```
if (a > b) {
    System.out.println("A es mayor que B?");
    System.out.println("Si, claro que A es mayor que B");
}
```

En Python el concepto de bloques se desprende de la estructura e indentación del programa. El ejemplo anterior, debe escribirse de la siguiente forma en Python.

```
if (a > b) :
<TABULADOR>print "A es mayor que B?";
<TABULADOR>print "Si, claro que A es mayor que B";
```

Donde <TABULADOR> representa un solo carácter de tabulación desde el comienzo del if. Luego de la sentencia que contendrá una serie de sentencias anidadas, deberá utilizarse el carácter ":", como indicador de que a continuación sigue una serie de sentencias anidadas a la sentencia que contiene los ":"

Estructuras de control

Se deben implementar como mínimo las siguientes estructuras de control.

If

```
fav = "mostrar_saludo"

if fav == "mostrar_saludo":
   print "Hola que tal"
   print "Bienvenidos"
```

If ... Else

```
if fav == "mostrar_saludo":
   print "Hola que tal"
   print "Bienvenidos"
else:
   print "Chau chau!"
```

While

```
edad = 0

while edad < 18:
   edad = edad + 1
   print "Felicidades, tienes " + str(edad)

print "Felicidades, ya es mayor de edad!"
```

Break

```
while True:
   entrada = raw_input("> ")
   if entrada == "adios":
      break
   else:
      print entrada
```

Continue

```
edad = 0

while edad < 18:
   edad = edad + 1
   if edad % 2 == 0:
      continue
   print "Felicidades, tienes " + str(edad)
```

For … In

```
secuencia = ["uno", "dos", "tres"]

for elemento in secuencia:
    print elemento
```

Funciones

En Python una función es un fragmento de código con un nombre asociado que realiza una serie de tareas y devuelve un valor. Cuando la función no especifica un valor de retorno, Python devuelve None (vacío) como valor de retorno. None es equivalente a NULL en otros lenguajes, como ser C o Java. Para definir una función, utilizamos la siguiente sintaxis:

```
def mi_funcion(param1, param2):
    sentencia 1
    sentencia 2
    ...
    sentencia N
```

Para realizar una invocación a una funcion, basta realizar cualquier de las siguientes llamadas:

```
mi_funcion(valor_param1, valor_param2)

mi_funcion(param1 = valor_param1, param2 = valor_param2)
```

En Python, el pasaje de parámetros siempre es realizado **POR VALOR**. En el caso de los tipos de datos primitivos, se realiza una copia del valor. En el caso de tipos complejos, como

colecciones, se realiza una copia de la dirección de memoria donde se encuentra dicha estructura.

Por lo tanto, si se quiere realizar un pasaje de un parámetro por referencia, entonces se debe colocar el mismo dentro de una lista, y pasar la lista como parámetro.

Para devolver valores, se utiliza la palabra clave return. Esta puede devolver un solo valor, o un conjunto de valores a ser asignados en un conjunto de variables.

```
def sumar(x, y):
    return x + y

print sumar(3, 2)
```

```
def f(x, y):
    return x * 2, y * 2

a, b = f(1, 2)
```

Funciones predefinidas

Para los diferentes tipos de datos antes presentados, así como para la operativa básica del lenguaje, se deberán soportar las siguientes funciones predefinidas.

- Para Diccionarios
    - D.has_key(K), indica si el diccionario contiene la clava K
    - D.items(), devuelve una lista de tuplas con las parejas clave-valor del diccionario
    - D.keys(), devuelve la lista de las claves del diccionario
    - D.pop(K), elimina el valor asociado a la clave K
    - D.values(), devuevle la lista de los valores del diccionario

- Para Strings
    - S.count(sub), cuenta las ocurrencias de la subcadena en S
    - S.find(sub), localiza la subcadena en S
    - S.find(sub, start), localiza la subcadena en S, pero comenzando en start
    - S.join(secuencia), devuelve una nueva cadena, resultado de concatenar los elementos de la secuencia, separados por la cadena S
    - S.split(sep), devuelve una lista de cadenas, resultado de separar la cadena S con el separador sep
    - S.replace(old, new), reemplaza la primer ocurrencia de la subcadena old en S, por la subcadena new
    - S.length(), retorna el largo de la cadena S

- Para Listas
    - L.append(ítem), agrega el ítem indicado a la lista L
    - L.count(value), cuenta las ocurrencias de value en L

- L.extend(lista), agrega todos los ítems de la lista, a L
- L.index(value), devuelve el primer indica de value en la lista L
- L.index(value, start), similar al anterior, pero comenzando en la posición start
- L.insert(index, value), inserta el elemento value, en el índice index
- L.pop(index), elimina el valor en la posición L
- L.size(), devuelve el tamaño de la lista L

Entrada / Salida

Se deberán soportar las siguientes funciones para el manejo de la entrada y salida en el intérprete de Python a construir.

Para leer un dato desde la entrada estándar, se podrá usar la función raw_input(). Esta recibe un string que se despliega, y luego espera que se ingrese un valor y se presione ENTER. Es equivalente a un readline() en otros lenguajes.

```
nombre = raw_input("Como se llama? ")

print "Hola que tal!, " + nombre
```

Para mostrar un mensaje por la salida estándar, basta con utilizar la función print(), la cual muestra el mensaje indicado, en una nueva línea en la salida estándar.

```
print "Hola mundo!"

>>> Hola mundo!
```

Conversión de tipos

Existen diversas funciones en Python para convertir elementos de un tipo de datos a otro. Las funciones a continuación reciben un elemento X, con un tipo de dato valido en Python, debiendo convertirlo al tipo de datos indicado en la función. El caso más común es que X sea un valor de tipo String. Sin embargo, es posible que se realicen otro tipo de conversiones, por ejemplo entre enteros y flotantes.

- int(X), convierte el valor X a entero
- float(X), convierte el valor X a flotante
- str(X), convierte el valor X a string
- tuple(X), convierte el valor X a una tupla
- list(X), convierte el valor X a una lista
- dict(X), convierte el valor X a un diccionario

3. Aspectos de implementación

A continuación se presentan algunos de los aspectos a tener en cuenta en la implementación del intérprete de Python indicado.

Lenguaje

El lenguaje presentado en la sección 2 pretende ser una guía para el desarrollo del intérprete. Los aspectos presentados representan los requerimientos mínimos incluir en la solución. En caso de que algún aspecto no haya sido considerado, pero sea claramente de sentido común incorporarlo, esto puede ser realizado sin inconvenientes. En cualquier caso, consulte con el docente del curso si es posible agregar dicha funcionalidad.

Ambiente de ejecución

Se deberá implementar un ambiente de ejecución apropiado para permitir ejecutar los programas escritos en el lenguaje solicitado. Entre otras cosas, este ambiente de ejecución regulara el llamado a funciones, la gestión de memoria, manejo de variables, registros de activación, etc.

Tratamiento de errores

El requerimiento mínimo es terminar el proceso de interpretación una vez que se detecta un error indicando el número de línea y de ser posible el tipo de error (sintáctico, de tipo, etc.).

Son aceptables soluciones en los que los mensajes de error son de la forma:

- `Error sintáctico en línea 2 cerca de "x = 10"`
- `Tipo de datos incorrecto en sentencia IF en línea 10`

Mecanismos para el manejo de error más elaborados, serán tenidos en cuenta, pero se sugiere comenzar con un manejo mínimo del error, y luego expandirlo si se tiene tiempo disponible.

Programas de prueba

Se deberán entregar programas de prueba que verifiquen el correcto funcionamiento de interprete, testeando aspectos de análisis léxico, sintáctico, semántico y de interpretación de los programas Python.

4. Objetivos, formas y plazos de entrega

Algunos aspectos formales a tener en cuenta…

Sobre el objetivo

El objetivo principal del obligatorio es desarrollo de un intérprete del lenguaje Python presentado en la sección 2, que FUNCIONE.

Las características del lenguaje que deben ser tenidas en cuenta para dicho intérprete son las presentadas en este documento. Toda otra característica que no esté en este documento, se considera opcional (a menos que sea aclarada su obligatoriedad en la página web del curso)

Sobre los grupos

El trabajo deberá ser realizado en grupos de hasta **tres estudiantes**. Los mismos deberán ser informados antes del 12/6 al docente encargado del curso. El día 15/6 será informada la lista final de grupos del obligatorio. Se considerar que los alumnos que no hayan formado grupo antes de la fecha indicada, trabajaran en forma individual en el obligatorio.

IMPORTANTE: Una vez formado un grupo, no se aceptaran separaciones o divisiones en dicho grupo. En caso de darse esta situación, TODOS los integrantes del curso pierden el obligatorio. En caso de que parte de los integrantes abandonen, solo a estos se les considera como no aprobado el curso.

Sobre las consulta

Se realizaran clases de consulta los días miércoles, hasta la fecha de entrega, en el horario de 8:30 a 9:30, en el salón 501. En cualquier caso, se pueden enviar consultas al docente encargado del curso, a la dirección **pgarbusi@fing.edu.uy**.

Sobre la plataforma de trabajo

El trabajo deberá ser realizado utilizando el lenguaje Java. Se deberán utilizar los utilitarios JFLEX y CUP para realizar el análisis léxico y sintáctico respectivamente. En caso de ser necesario, se podrá utilizar código descargado de Internet (debidamente documentado), siempre y cuando este NO RESUELVA una parte fundamental del problema planteado. Por ejemplo, no sería aceptable si se entrega un intérprete de PYTHON no desarrollado por el grupo.

Las pruebas de correctitud del obligatorio serán realizadas en plataforma Windows. Puede desarrollar en cualquier plataforma que considere necesaria, pero se recomienda probar en plataforma Windows lo que se va a entregar.

MUY IMPORTANTE:

Se deberá entregar el código fuente de la solución, así como los scripts necesarios para compilar y obtener una versión funcional del intérprete. El grupo se deberá asegurar que el código fuente se puede compilar, ya que SOLO se ejecutara el intérprete obtenido de la compilación de las fuentes entregadas. Al ser el obligatorio desarrollado en Java, puede utilizarse ANT o MAVEN para automatizar el proceso.

No se aceptaran soluciones que utilicen archivos propietarios de proyectos desarrollados en IDEs como Eclipse, JBoss Studio, Websphere, etc. El grupo deberá asegurarse que el código se puede compilar sin necesidad de contar con dichas IDEs.

Se recomienda entregar el código fuente junto con un script que genere el binario del intérprete.

Sobre la entrega

La fecha de entrega final para este obligatorio es el viernes 17 de julio del 2015 (inclusive)

Se deberá entregar al docente del curso, en forma electrónica, lo siguiente:

- **BREVE** documentación del obligatorio presentando los aspectos principales del diseño de la gramática, chequeos, las tablas de símbolos, la representación intermedia, tratamiento de error y el proceso de interpretación. Se espera, muy especialmente, que fundamenten adecuadamente las decisiones de diseño tomadas a lo largo del obligatorio.

- Los archivos de prueba ejecutados por el grupo sobre el intérprete. En caso de las pruebas que contienen error, indicar los errores obtenidos. En caso de que haya un problema conocido con la ejecución de los programas de prueba entregados, esto deberá ser documentado apropiadamente.

Sobre la evaluación

Se evaluaran los siguientes puntos:

- **Correcto funcionamiento del intérprete**, ejecutando los programas de prueba entregados por el grupo, en los cuales se prueben los aspectos del lenguaje que deben ser implementados. **SI EL INTERPRETE NO FUNCIONA CORRECTAMENTE, EL CURSO SE CONSIDERA PERDIDO PARA TODOS LOS INTEGRANTES**

- **Calidad en la implementación**. **SOLO SI EL INTERPRETE FUNCIONA CORRECTAMENTE**, se evaluaran como puntos extra, la calidad con que sea haya implementado la solución, las características extra que se le hayan agregado al intérprete, las mejoras al manejo e informe de errores, etc.

- En caso de duda por parte del docente sobre la implementación del interprete, los integrantes del grupo (en su totalidad o parcialmente) podrán ser llamados a una defensa, en la cual se les puede realizar preguntas concretas sobre la implementación realizada. **EN CASO DE COMPROBARSE QUE UN MIEMBRO DEL EQUIPO NO TRABAJO EN EL OBLIGATORIO, EL CURSO SE CONSIDERA PERDIDO PARA DICHO INTEGRANTE**

- El puntaje máximo del obligatorio es de 100 puntos. Se aprueba con 60 puntos. Este puntaje se promedia con el obtenido en la parte teórica del curso para forma la nota final del curso.

Sobre las copias

Para resolver este tema, se utilizara el mecanismo tradicional aplicado en el InCo cuando se detectan copias entre los grupos**. TODOS LOS INVOLUCRADOS PIERDEN EL CURSO** y se informa apropiadamente a la Facultad sobre esta situación.

Referencias

"Python para todos"
https://launchpadlibrarian.net/18980633/Python%20para%20todos.pdf