INF-253 Lenguajes de Programación
Tarea 1: Python

Profesor: Esteban Daines - Roberto Diaz
Ayudante Cátedras: Sebastian Godinez
Ayudante Tareas: Gabriel Valenzuela - Monserrat Figueroa

10 de Septiembre de 2018

1. Objetivos

El alumno aplicará conceptos de **Expresiones regulares** para realizar un programa que analice gramaticalmente programas simples.

2. Especificaciones del Problema

LOLCODE es un lenguaje esotérico inspirado en *memes* de gatos que rondan por el internet. Un grupo de programadores que se vio obligado a trabajar con este lenguaje ha tenido muchos problemas al no poder notar los errores fácilmente, ya que éste no requiere identación o uso de caracteres especiales (como paréntesis) para ordenar el código. Por lo tanto, para ayudarlos se le pidió a los estudiantes de Lenguajes de Programación que implementen un programa el cual permita identificar la estructura de las sentencias escritas y los errores de sintaxis presentes en el código.

3. Requerimientos

- El alumno debe utilizar Python 3 para el desarrollo de esta tarea.
- Se recomienda trabajar en terminales que usen comandos UNIX como los computadores del LDS.
- El programa debe utilizar expresiones regulares para el análisis de sentencias. Pueden consultar la documentación del módulo **re** en el siguiente enlace.
- El programa debe mostrar por consola el código con colores utilizando códigos de escape ANSI. Para mayor información sobre tal característica puede ingresar en el siguiente enlace.

4. Introducción a LOLCODE

4.1. Estructura básica

A continuación se presentan las diferentes estructuras sintácticas presentes en el lenguaje **LOLCODE** que deberán ser revisadas por su programa:

Sintaxis

```
HAI
<Codigo a ejecutar>
KTHXBYE
```

La existencia de lineas de código antes de *HAI* o después de *KTHXBYE* significan un error en la sintaxis del programa.

4.2. Declaración e inicialización de variables

Palabras claves

- I HAS A: Declaración
- ITZ: Inicialización
- R: Asignación

Sintaxis

```
I HAS A <nombre_variable> [ITZ <expresion>]
<nombre_variable> R <expresion>
```

Es importante notar que la inicialización y la asignación son cosas diferentes; la inicialización puede o no estar en la misma linea de la declaración, mientras que la asignación se realiza (o no) en alguna parte posterior del código. Por otro lado, los nombres de las variables deben cumplir con las siguientes condiciones:

- Se pueden utilizar letras minúsculas y mayúsculas
- Puede contener números y guiones bajos (_), pero debe comenzar con letras

4.3. Operadores

4.3.1. Matemáticos

- SUM OF: suma de dos valores
- DIFF OF: resta de dos valores
- PRODUKT OF: multiplicación de dos valores
- QUOSHUNT OF: división de dos valores

- MOD OF: modulo
- BIGGR OF: cual es el mayor de dos valores
- SMALLR OF: cual es el menor de los valores

4.3.2. Booleanos

- BOTH OF: Conjunción Lógica (AND)
- EITHER OF: Disyunción Lógica (OR)
- NOT: Negación Lógica

4.3.3. Comparacion

- BOTH SAEM: Igualdad
- DIFFRINT: Diferencia

Sintaxis

Binarios:

```
<operator> <expression1> AN <expression2>
```

Unarios:

```
<operator> <expression>
```

4.4. Condicionales

Palabras claves

- O RLY?: Se cumple lo anterior? (Pregunta de apertura)
- YA RLY: Si sucede
- NO WAI: No sucede
- OIC: Oh, ya veo (Fin de la sentencia)

Sintaxis

```
<Expresion>
O RLY?
YA RLY
<Codigo que se ejecuta si se cumple la expresion>
NO WAI
<Codigo que se ejecuta si no se cumple la expresion>
OIC
```

4.5. Loops

- TIL: hasta
- WILE: mientras

Sintaxis

```
IM IN YR <nombre del loop> <operacion> YR <variable>...
...[TIL|WILE <expression>]
   <bloque de codigo que se ejecuta en el loop>
IM OUTTA YR <nombre del loop>
// La expresion "IM IN YR..." se encuentra en una sola linea
//<operacion>:=NERFIN(disminuir por 1)|UPPIN(aumentar por 1)
```

4.6. Entrada y Salida

- GIMMEH: Entrada
- VISIBLE: Salida

Sintaxis

```
GIMMEH <nombre_variable>
VISIBLE <expresion>
```

Para mayor información sobre el lenguaje puede visitar la documentación oficial. Sin embargo, sólo se evaluará la sintaxis presentada en este documento.

5. A implementar

Para implementar el verificador de sintaxis se utilizarán códigos de escape ANSI, los cuales permiten dar formato a la salida de texto por consola, además del uso de expresiones regulares. El programa debe recibir un nombre de archivo por consola, el cual contendrá el código a ser verificado para, posteriormente, imprimir su contenido con las palabras claves destacadas haciendo uso de los colores correspondientes. En caso que exista errores en la sintaxis, deben destacarse las lineas que generan los errores.

Colores a utilizar

- Azul: Operadores (incluye 'AN')
- Cyan: Condicionales
- Purpura: Loops
- Verde: HAI y KTHXBYE

- Amarillo: Declaración e inicialización de variables
- Rojo: Asignación de variables, Entrada y Salida de datos
- Rojo: Errores de sintaxis

Considere que los casos de expresiones que contienen una sentencia de apertura y de finalización (como por ejemplo O RLY? y OIC) deben poseer ambos términos para que éstas sean válidas. En caso contrario (es decir, que falte alguna de las dos expresiones), la sentencia presente en el código debe destacarse con rojo.

6. Ejemplos

Cálculo de Factorial

- Entrada correcta

```
HAI
I HAS A num ITZ 5
I HAS A cont ITZ 0
I HAS A aux ITZ 1
IM IN YR ciclo UPPIN YR cont TIL BOTH SAEM cont AN num
aux R PRODUKT OF aux AN SUM OF cont AN 1
IM OUTTA YR ciclo
VISIBLE aux
KTHXBYE
```

Salida

```
HAI
I HAS A num ITZ 5
I HAS A cont ITZ 0
I HAS A aux ITZ 1
IM IN YR ciclo UPPIN YR cont TIL BOTH SAEM cont AN num
aux R PRODUKT OF aux AN SUM OF cont AN 1
IM OUTTA YR ciclo
VISIBLE aux
KTHXBYE
```

- Entrada con Errores

```
HAI
I HAS A num ITZ 5
I HAS A cont ITZ 0
I HAS A aux ITZ 1
IM IN YR ciclo UPPIN YR cont TIL BOTH SAEM cont AN num
aux R PRODUKT OF aux AN OF SUM cont AN 1
VISIBLE aux
KTHXBYE
```

Salida

```
HAI
I HAS A num ITZ 5
I HAS A cont ITZ 0
I HAS A aux ITZ 1
IM IN YR ciclo UPPIN YR cont TIL BOTH SAEM cont AN num
aux R PRODUKT OF aux AN OF SUM cont AN 1
VISIBLE aux
KTHXBYE
```

7. Archivos a Entregar

- checker.py
- README.txt

8. Sobre Entrega

- **Se debe trabajar en grupos de DOS personas.**
- La entrega debe realizarse en tarball (tar.gz) y debe llevar el nombre: "Tarea1LP_RolIntegrante-1_RolIntegrante-2.tar.gz". **Tareas que no cumplan esta regla llevarán descuento**.
- El archivo README.txt debe contener nombre y rol de los integrantes del grupo e instrucciones **claras** para la utilización de su programa. Aquí pueden agregar también distintos supuestos que hayan utilizado para el desarrollo (conversado previamente con los ayudantes). **Tareas que no cumplan esta regla llevarán descuento**.

- Su código debe estar comentado de tal forma que sea comprensible. Éstos deben venir antes del código que comentan y, en caso de funciones, debe especificar los parámetros, los valores de retorno y una breve descripción de lo que hace. **Tareas que no cumplan esta regla llevarán descuento**.
- Todas las dudas deben ser hechas por la plataforma *Moodle*.
- La entrega será mediante *Moodle* y el plazo máximo de entrega es hasta el Domingo 30 de Septiembre a las 23:55.
- Por cada día o fracción de atraso se descontarán **20 puntos**. Si la fracción es menor a 20 minutos, el descuento será de sólo **5 puntos**.
- Las copias serán evaluadas con nota 0.

9. Calificación

La escala a utilizar para la revisión de la tarea es la siguiente:

- Código (80 Puntos)
  - Orden (5 Puntos)
  - Expresiones Regulares (25 Puntos)
  - Implementación (50 Puntos)
- Consola (20 Puntos)
  - Se muestran los colores correctamente (10 Puntos)
  - Se identifican los errores (10 Puntos)

Los descuentos que pueden ocurrir son:

- No uso de Expresiones regulares (100 puntos)
- No respetar reglas de entrega (40 puntos)