Teoría de Lenguajes
Primer Laboratorio – 2020

El propósito de este laboratorio es trabajar con expresiones regulares de una manera práctica, para lo cual se propone escribir programas en el lenguaje Python.

El trabajo está enfocado en resolver los problemas haciendo un fuerte uso de las expresiones regulares, evitando sustituirlas por sentencias de programación. Además, deben usarse expresiones regulares lo más genéricas posible e independientes de datos de entrada particulares. En general los programas se pueden resolver en menos de 15 líneas.

Modo de Trabajo

- Se indicarán 5 programas a realizar.

- Se entregará para cada programa: un archivo de entrada y un archivo con la salida de referencia que contiene la salida que se debería obtener para el archivo de entrada correspondiente.

- El estudiante, en base a lo especificado por la letra, la entrada y la correspondiente salida de referencia deberá realizar su programa.

- La salida del programa del estudiante deberá ser idéntica a la salida de referencia. Algunos detalles de cómo debe trabajar su programa surgen de analizar la entrada y su correspondiente salida. Para las comparaciones se entregará el programa “diff.exe” para Windows que, tomando como entrada dos archivos, detecta si hay diferencias.

- Se deberá respetar los nombres de los archivos de entrada, de salida, y de los programas Python.

- Se proporcionará un archivo comprimido que contiene lo siguiente:
    - los archivos de entrada.
    - los archivos con las salidas de referencia.
    - los archivos Python que deben implementar, dentro del subdirectorio “programas”.
    - un script para facilitar las tareas de ejecutar y comparar las salidas (test.py).
    - el programa diff.exe para comparar las salidas.

Formato de las Entradas

Las entradas serán funciones escritas en un lenguaje de programación ficticio, con las características descritas a continuación.

Se puede asumir que las entradas consistirán de una única función, opcionalmente con comentarios.

Tipos de datos:

- **Entero**
- **Real**
- **Lista<_>**
- **Booleano**

Estructuras de control:

- **si-sino**: funciona como el *if-else.*
- **para-cada**: funciona como el *foreach.*
- **para**: funciona como el *for.*
- **mientras**: funciona como el *while.*

Operaciones no booleanas:

- Suma (**+**)
- Resta (**-**)
- Multiplicación (*****)
- División (**/**)

Operaciones booleanas:

- Mayor (**>**)
- Menor (**<**)
- Mayor o igual (**>=**)
- Menor o igual (**<=**)
- Igual (**==**)
- And lógico (**&&**)

Comentarios

- Comentario de una línea: se marca su comienzo con " // " y ocupa toda la línea.
- Comentario multilínea: se marca tanto su comienzo como su final con " *** ", y puede (o no) ocupar varias líneas.

Funciones

- La única función que provee el lenguaje es **largo()** , que devuelve el largo de una lista.

Otras palabras reservadas:

- **funcion**: se utiliza al declarar una función.
- **devolver**: funciona como el *return*, es decir que se usa para finalizar la ejecución de una función y devolver un valor.

Restricciones del lenguaje:

- Solo puede haber una única sentencia por línea.
- Los nombres de variables comienzan con una letra minúscula, y solamente pueden contener letras, números y guiones bajos.
- En una línea no puede haber código y comentarios al mismo tiempo (los comentarios deben estar en líneas en las cuales no haya código).
- Las palabras reservadas se escriben en minúscula.

El resto de la sintaxis del lenguaje se puede deducir observando los ejemplos de entradas que se presentarán a continuación.

programa1.py

Dada una función en el lenguaje descrito, despliega el mismo código pero sin comentarios.

Al remover un comentario, no debe quedar una línea en blanco, pero de haber líneas en blanco desde antes, éstas deben permanecer en el código.

Ejemplo de entrada:

```
***Calcula en indice de masa corporal***
funcion calcularIMC(alturaEnMetros: Real, pesoEnGramos: Real) -> Real {
      // Hay que pasar a kilogramos para aplicar la ecuacion
      pesoEnKilos: Real = pesoEnGramos / 1000
      alturaAlCuadrado: Real = alturaEnMetros * alturaEnMetros
      imc: Real = pesoEnKilos / alturaAlCuadrado

      devolver imc
}
```

Ejemplo de salida :

```
funcion calcularIMC(alturaEnMetros: Real, pesoEnGramos: Real) -> Real {
      pesoEnKilos: Real = pesoEnGramos / 1000
      alturaAlCuadrado: Real = alturaEnMetros * alturaEnMetros
      imc: Real = pesoEnKilos / alturaAlCuadrado

      devolver imc
}
```

programa2.py

Dada una función en el lenguaje descrito, despliega un resumen con la cantidad de sentencias de control (*si*, *para*, *para-cada* y *mientras*) que contiene.

Ejemplo de entrada :

```
***Si la matriz es simetrica, devuelve verdadero, sino devuelve falso***
funcion es_simetrica(M: Lista<Lista<Entero>>) -> Booleano {
      i: Entero = 1
      resultado: Booleano = verdadero
      // Uso mientras para cortar antes la iteracion si ya se que no es simetrica
      mientras resultado && i <= largo(M) {
          j: Entero = i + 1
          mientras resultado && j <= largo(M(i)) {
                    si M(i)(j) == M(j)(i) {
                         j = j + 1
                    } sino {
                         resultado = falso
                    }
          }
                 i = i+1
      }
      devolver resultado
}
```

Ejemplo de salida:

```
si: 1
para: 0
para-cada: 0
mientras: 2
```

programa3.py

Dada una función en el lenguaje descrito, despliega la siguiente información sobre su cabezal:

- En la primer línea, el nombre de la función.
- En la segunda línea, sus parámetros de entrada junto a sus tipos, separados por comas.
- En la tercer línea, el tipo de la salida.

Si la función no tiene salida o parámetros de entrada, la línea correspondiente no se imprime.

Ejemplo de entrada:

```
funcion interes_compuesto(capital_incial: Real, intereses_anuales: Lista<Real>) -> Real {
      capital_final: Real = capital_inicial
      anios: Entero = largo(intereses_anuales)
      para i: Entero = 0 -> anios - 1 {
          // Aplico el interes del anio
          capital_final = capital_final * intereses_anuales[i]
      }
      devolver capital_final
}
```

Ejemplo de salida:

```
interes_compuesto
capital_incial: Real, intereses_anuales: Lista<Real>
Real
```

programa4.py

Dada una función en el lenguaje descrito, despliega la cantidad de operaciones booleanas y no booleanas que tiene.

Ejemplo de entrada:

```
***Si la matriz es simetrica, devuelve verdadero, sino devuelve falso***
funcion es_simetrica(M: Lista<Lista<Entero>>) -> Booleano {
      i: Entero = 1
      resultado: Booleano = verdadero
      // Uso mientras para cortar antes la iteracion si ya se que no es simetrica
      mientras resultado && i <= largo(M) {
          j: Entero = i + 1
          mientras resultado && j <= largo(M(i)) {
                    si M(i)(j) == M(j)(i) {
                         j = j + 1
                    } sino {
                         resultado = falso
                    }
          }
                 i = i+1
      }
      devolver resultado
}
```

Ejemplo de salida:

```
booleanas: 5
no booleanas: 3
```

programa5.py

Dada una función en el lenguaje descrito, despliega las variables que define, junto con sus valores iniciales.

Ejemplo de entrada:

```
funcion interes_compuesto(capital_incial: Real, intereses_anuales: Lista<Real>) -> Real {
      capital_final: Real = capital_inicial
      anios: Entero = largo(intereses_anuales)
      para i: Entero = 0 -> anios - 1 {
          // Aplico el interes del anio
          capital_final = capital_final * intereses_anuales[i]
      }
      devolver capital_final
}
```

Ejemplo salida:

```
(capital_final,capital_inicial)
(anios,largo(intereses_anuales))
(i,0)
```

Herramientas a utilizar

Se usará Python 3.8. Por más información ver instructivo.

Desarrollo del trabajo

Los trabajos se deben realizar en grupos de 2, 3 o 4 estudiantes. No puede realizarse en forma individual. Se sancionará con la pérdida del laboratorio si se detectan trabajos copiados, o si se hace una mala utilización de los foros de consulta.

Entrega

Se realizará vía web, y se comunicarán los detalles oportunamente. La misma se habilitará un par de días antes del vencimiento el día **martes 28 de abril hasta las 23:55 hs.**

Se entregarán los **5 programas**, cuyos nombres deberán ser exactamente los especificados (respetando mayúsculas, minúsculas, guiones, etc.).
Se entregará un archivo de texto de nombre **integrantes.txt** que contendrá la CI (7 dígitos) y el nombre de cada uno de los integrantes del grupo (sin usar tildes), y a continuación algún comentario que quieran hacer en el caso que algún programa no les hubiera funcionado correctamente. No necesitan inscribir el grupo previamente.

Formato del archivo *integrantes.txt*:

```
3123456,Gonzalez,Pablo
4567123,Martinez,Veronica
3444555,Garcia Rodriguez,Juan Jose
Comentarios (sólo si algún programa no hubiera funcionado bien).
```

Se debe respetar el formato del archivo:

- datos de los integrantes: “CI (7 dígitos sin guiones)” “coma” “Apellido/s” “coma” “Nombre/s”
- un integrante por línea, en las primeras líneas del archivo.

Corrección

Es parte de las normas usar los scripts entregados y los nombres de los programas indicados. Es importante cumplir la norma y se penalizará en la corrección si no se sigue adecuadamente.

La corrección se realizará usando los mismos scripts entregados para probar, por lo tanto si la ejecución cancela, o hay diferencia en los archivos de salida con la solución oficial, la solución no se considerará correcta.

Es importante, como ya se aclaró al principio, usar la funcionalidad de las expresiones regulares del lenguaje y no sustituirlas por sentencias de programación.

Referencias

Se recomienda leer las siguientes páginas para aprender sobre el uso de expresiones regulares en Python:

- https://docs.python.org/3/library/re.html
- https://www.w3schools.com/python/python_regex.asp