Universidad Simón Bolívar
Departamento de Computación y Tecnología de la información
CI-2691- Laboratorio de algoritmos I

Pre-Laboratorio 2

El objetivo del laboratorio 2 es traducir algoritmos dados en GCL a Python, estudio de condicionales, instrucción nula, indentación y documentación del código.

Contenido: Indentación, Documentación, Tamaños de Línea, Condicionales, Instrucción Nula.

Indentación:

En Python no se tienen paréntesis, corchetes o palabras claves, como *begin* o *end* para delimitar el comienzo y final del un programa. Tampoco se necesita delimitadores como punto y coma para finalizar las instrucciones. Lo que permite organizar los bloques de código es la indentación.

La indentación se refiere al espacio en blanco o sangría dejado al comienzo de la línea con el propósito de hacer más legible el código. En el caso de Python, la indentación además indica un bloque de instrucciones. Por ello, se debe agregar indentación cada vez que se abre un nuevo bloque o sección de código, es decir, una lista de instrucciones en secuencia dentro de una acción compuesta. Los editores que reconocen la sintaxis del lenguaje de programación suelen ayudar a tener una buena indentación. Si un conjunto de instrucciones está indentado con la misma sangría, esto forma un bloque con el mismo nivel hasta el fin de archivo o hasta que se encuentre una línea con menor indentación.

Hay varios opciones para indentación, pero el estándar es tener cuatro espacios o columnas, por nivel de indentación. A continuación se coloca un ejemplo de indentación:

```
if condición booleana del condicional:
      # Comienzo del Primer Bloque Interno
            if  condición booleana del condicional anidado:
                  # Segundo Bloque (dentro del condicional anidado)
                  print(x)
            # de vuelta al bloque anterior
            # Otras instrucciones del primer bloque.
      # Continuación del programa
```

Documentación:

Los comentarios deben ser usados para dar una visión del código y proporcionar información adicional que no se encuentra disponible en el mismo. Los comentarios sólo deben contener información necesaria para leer y comprender el programa. Por ejemplo, información sobre

cómo se debe compilar o ejecutar el programa o en que directorio se encuentra no debe ser incluida como comentario. Se debe incluir información acerca de decisiones de diseño del programa que no sean triviales o que no sean obvias al leer el código.

Debe tenerse el cuidado de no repetir información que ya esté presente en el código y pueda ser obtenida claramente a partir de éste.

Para colocar un comentario en la línea actual utilice el símbolo numeral, '**#**'.

Ejemplo:

```
# Esto es un comentario desde el comienzo de línea
x = 3     # esto es un comentario después del código.
```

Para comentarios más largos o para una documentación completa, especialmente al comienzo de un archivo .py, utilice la triple comilla doble. Todo lo que quede encerrado entre la triple comilla será ignorado por Python.

Ejemplo:

```
""" Un ejemplo de código en python.
      Note que las triple comillas permiten tener varias líneas y todas
      son ignoradas.
"""
x = 3
```

Para mayor detalle se recomienda revisar las referencias [1].

Condicionales:

En GCL, el condicional está precedido de la palabra `if` y a continuación una guardia o expresión booleana seguida del símbolo "`->`", después se coloca el bloque de instrucciones internas al `if` correspondientes a esa guardia. El `if` finaliza con la palabra `fi`. Es posible colocar varias guardias usando el símbolo "`[]`" para preceder el siguiente grupo de "guardia `->` instrucciones". En GCL es importante garantizar que las guardias están completas, es decir, que la disyunción de estas produzca el universo de todos los casos posibles.

El capítulo 4 del manual de Python trata los condiciones, los cuales se introducen aquí con un ejemplo:

```
if x < 0:
      x = 0
      print('Negativo convertido en cero')
elif x == 0:
      print('Cero')
```

```
elif x == 1:
      print('Uno')
else:
      print('Más de uno')
```

En este condicional, observamos que comienza con la palabra reservada `if` seguida de una expresión booleana que termina con el símbolo de continuación "`:`" (que sustituye al símbolo "`->`" de GCL). El bloque de instrucciones a realizar cuando la expresión es cierta van en la línea siguiente, indentadas a la derecha con al menos un espacio adicional al comienzo del `if`. Sabremos que las acciones se terminan cuando se regrese al nivel de indentación del `if`. Si hay más condiciones o guardias a verificar, se regresa al nivel de indentación del `if` y se inicia la siguiente guardia con la palabra reservada `elif` (que sustituye al símbolo "`|`" de GCL) seguida de una expresión booleana, de los dos puntos, y en la línea siguiente el bloque de instrucciones (indentadas) correspondientes a esta segunda guardia. Se puede agregar tantas guardias como sea necesario.
Para garantizar que se cubrieron todos los casos posibles (completitud de las guardias) se regresa al nivel de indentación de `if` y se escribe la palabra reservada `else` seguida de dos puntos y en la línea siguiente las instrucciones correspondientes a este caso. Las secciones `elif` y `else` son opcionales. Note que en Python no es necesario un delimitador final como el `fi` de GCL pues basta con regresar la indentación al nivel del `if` para indicar que éste terminó.
Para concretar, el **if** se utiliza para establecer una condición en un programa, por ejemplo si **A >=1** (A mayor o igual a 1), el **elif** sería la una condición intermedia y contraria a la primera condición (**A >=1**) que para este caso sería que **A**<**0** (A menor que 0), y el **else** es la condición contraria y por defecto al **if** y al **elif**, por ejemplo si no se cumple con A >=1 y tampoco se cumple A<0**,** la condición que complementa el universo de posibilidades sería **A**=**0** (A igual a 0).
A continuación se presenta la equivalencia utilizando **if** y **else** entre los lenguajes GCL y Python.

GCL	Python

`if` guardia1 `->`
bloque_instrucciones1
`|` `not` guardia1 `->`
bloque_instrucciones2
`fi;`
siguiente_instrucción
	`if` guardia1 **`:`**
	bloque_instrucciones1
	`else :`
	bloque_instrucciones2
	siguiente_instrucción

Ejemplo 1: Dado un numero entero, determinar si par o impar

GCL	Python

if (numero mod 2)=0 ->
  write("El numero es par");
| not (numero mod 2)=0 -> write("El numero es impar");
fi;
	# la función int convierte a entero un string
	numero = **int**(input('Introduzca valor del numero: '))
	if numero % 2 == 0:
	  print ('El numero es par')
	else:
	  print ('El numero es impar')

Es necesario enfatizar que **elif** no existe en GCL, la otra forma de ver la equivalencia de GCL es como esta tabla, que traduce una estructura de selección **if**, **elif** y **else** de GCL a Python.

GCL	Python
if guardia1 -> bloque_instrucciones1 | guardia2 -> bloque_instrucciones2 | guardia3 -> bloque_instrucciones2 fi; siguiente_instrucción	**if** guardia1 : bloque_instrucciones1 **elif** guardia2 : bloque_instrucciones2 **elif** guardia3 : bloque_instrucciones2 **else** : **assert**(False) siguiente_instrucción

Ejemplo 2: Si gasto hasta $100, pago con dinero en efectivo. Si no, si gasto más de $100 pero menos de $300, pago con tarjeta de débito. Si no, pago con tarjeta de crédito.

GCL	Python

if compra <= 100 -> write("pago en efectivo");
| compra>100 || compra<300 -> write("Pago con tarjeta de débito");
| compra >= 300 -> write("Pago con tarjeta de crédito");
fi;
	# la función int convierte a entero un string
	numero = **float**(input('Introduzca el monto de la compra: '))
	if compra <= 100: **print** ("Pago en efectivo")
	elif compra > 100 and compra < 300: **print** ("Pago con tarjeta de débito")
	else: **print** ("Pago con tarjeta de crédito")

Para mayor detalle revisar el manual de Python [3].

Instrucción Nula:

La instrucción nula (skip de GCL) es útil cuando se combina con el condicional. En Python la instrucción nula corresponde a la palabra reservada **`pass`**. Esta instrucción simplemente no hace nada. Puede ser utilizada cuando se requiere colocar al menos una instrucción desde el punto de vista sintáctico, pero el programa no requiere realizar ninguna acción.

Si en el intérprete de Python se coloca el siguiente comando:

```
>>> pass
>>>
```

Observará que nada ocurre.

Para mayor detalle revisar [3].

Tamaños de Línea:

Es recomendable que todas las líneas queden a un máximo de 85 caracteres, o que por lo general éstas tienden a hacer la lectura del programa complicada y al imprimir el código se dificulta mucho su lectura.

La forma preferida de dividir líneas largas es utilizar la característica de Python de continuar las líneas de forma implícita dentro de paréntesis, corchetes y llaves. Si es necesario, puedes añadir un par de paréntesis extra alrededor de una expresión, pero algunas veces queda mejor usar una división invertida. Asegúrate de indentar la línea siguiente de forma apropiada. A continuación se coloca un ejemplo que muestra cómo dividir las líneas de diferentes maneras:

```
if size=='large':
       if width == 0 and height == 0 and \
              color == 'red' and emphasis == 'strong' or \
              highlight > 100:
              print("Has perdido")
       if width == 0 and height == 0 and (color == 'red' or
                                  emphasis == 'None'):
              print("Has ganado")
```

Se sugiere revisar directamente la guía de estilo citada en las referencias [2], en donde se explican convenciones y recomendaciones para tener una mejor calidad de código.

Ejercicios a entregar:

En los ejercicios 1 y 2. se les dan las especificaciones de dos programas, escritos en GCL. Debe escribir el programa equivalente en Python y llamarlo como indica cada uno de los ejercicios. Cree un archivo comprimido del tipo “tgz” llamado PreLab2-X.tgz, donde X es su número de carné, que contenga los programas Prelab2ejercicio1.py y Prelab2ejercicio1.py. Debe subir el archivo en el aula virtual, en la sección del Laboratorio 2, antes del martes 21 de Abril del 2015 a las 9:00 am.

1. **Prelab2ejercicio1.py**: Algoritmo que calcula el **valor absoluto de** “**a”**. No utilice la función `abs` de Python en cuerpo del programa, la puede usar para la postcondición.

```
“””
const a : int;
var   b : int; “””
{true}
# Algoritmo valor absoluto
{ b = abs(a) }
```

2. **Prelab2ejercicio2.py**: Algoritmo para calcular las **raíces del polinomio “AX^2+BX+C”**

```
“””
const A : int;
const B : int;
const C : int;
var x1 : float;
var x2 : float; “””

{A != 0 /\ 4 * A * C <= B * B}

# Algoritmo para calcular las raíces del polinomio AX^2 + BX + C

{ (A * x1 * x1 + B * x1 + C = 0 /\ A * x2 * x2 + B * x2 + C = 0) }
```

Referencias:

[1] DZone Refcardz, Core Python, By Naomi Ceder and Mike Driscoll. Disponible en Aula virtual con el nombre: dzone_refcardz.pdf.

[2] Guía de estilo del código Python, 10 de Agosto de 2007, Disponible en la web: http://mundogeek.net/traducciones/guia-estilo-python.htm

[3] The Python Standard Library, Document version 3.3, capítulo 4, Disponible en la web: https://docs.python.org/3.3/tutorial/controlflow.html