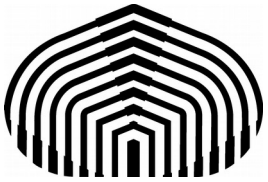

Universidad Simón Bolívar
Departamento de Computación y Tecnología de la información
CI-2691- Laboratorio de algoritmos I

Laboratorio 1

El objetivo del laboratorio es traducir especificaciones de entrada y salida a Python y verificar que los programas con asignación y secuenciación las cumplen. La plataforma en que se va a trabajar en este curso es GNU/Linux y el lenguaje de programación es Python 3. Los puntos a tratar en este laboratorio son: Instalación de Python 3 en GNU/Linux. Instalación del Editor recomendado en GNU/Linux. Familiarización con el ambiente de trabajo. Expresiones. Variables y asignación. Valores reales y complejos. Valores Booleanos. Comparadores. Cuantificadores acotados y predicados. Precondiciones y postcondiciones.

1. Instalación de Python 3 en GNU/Linux.

La instalación en GNU/Linux depende de la distribución que se esté utilizando. A continuación se indica el comando que debe usarse para instalar Python en Ubuntu, Arch Linux y Gentoo. En todas las distribuciones debe colocarse el comando indicado en un terminal y luego la clave cuando sea solicitada.

Comando en Ubuntu:
```
sudo apt-get install python3
```

Arch Linux:
```
sudo pacman -S python
```

Gentoo:
```
sudo emerge -av python
```

Recuerde tener la lista de repositorios actualizada antes de instalar el paquete. Para actualizar los repositorios, ejecute los siguientes comandos dependiendo de la distribución que utilice:

Ubuntu: Instale o actualice a la última versión de Ubuntu. Descárguela desde la página oficial. http://www.ubuntu.com/download/desktop . Y luego utilizar el siguiente comando:
```
sudo apt-get update
```

Arch Linux: Al actualizar el sistema, el gestor de paquetes automáticamente actualiza los repositorios. Para actualizar el sistema utilice el siguiente comando:
```
pacman -Syu
```

Gentoo: Utilice el siguiente comando:

```
emerge --sync
```

2. Instalación del Editor recomendado en GNU/Linux.

Para escribir programas en Python se recomienda usar los editores *gedit* o *geany*. A continuación se indica el comando que debe usar para instalar el editor *geany* en las principales distribuciones. Recuerde que esto debe hacerlo desde un terminal.

Comando en Ubuntu:

```
sudo apt-get install geany
```

Arch Linux:

```
sudo pacman -S geany
```

Gentoo:

```
sudo emerge -av dev-util/geany
```

A continuación se muestra el comando que debe usar para instalar el editor *gedit* en Ubuntu y Arch

Comando en Ubuntu:

```
sudo apt-get install gedit
```

Arch Linux:

```
sudo pacman -S gedit
```

3. Familiarización con el ambiente de trabajo - Comandos Shell

Para ejecutar los programas en *Python* y para desenvolverse en un ambiente Linux es necesario saber los comandos básicos para la administración en sistema tipo Unix. Los comandos son ejecutados en una aplicación llamada *Terminal*. Los comandos básicos que se necesita usar en un T*erminal* son los siguientes:

COMANDO	DESCRIPCIÓN
`pwd`	Da la ruta del directorio actual
`ls`	Lista los elementos del directorio actual
`cd [ruta/directorio]`	Permite cambiar de directorio
`cd ..`	Permite retroceder al directorio padre
`mkdir [directorio]`	Crea un directorio en la ruta actual
`rm -r [directorio]`	Elimina un directorio
`rm [nombre_del_archivo]`	Elimina un archivo
`clear (Ctrl+l)`	Limpia la pantalla
`python3 o python`	Ejecuta el interpretador de Python
`python3 [archivo.py]`	Corre un script en Python
`./[archivo.py]`	Ejecuta un script Python (Recuerde agregar los permisos de ejecución)
tecla Tabulador	Permite autocompletar
teclas de flechas UP, DOWN	Visualiza los comandos realizados
`Ctrl+C`	Detiene el proceso actual
`chmod +x archivo.py`	Le da permisos de ejecución a un archivo.

Ejercicios: Usando estos comandos realice las siguientes acciones desde un terminal:

1. Directorios (creación, eliminación, cambio)
    - Cree un directorio con su nombre para sus programas
    - Cámbiese al directorio creado.

2. Editor de código fuente (gedit, geany, emacs u otro)
    - Abra un algún editor de texto y cree un archivo en algunos de los directorios creados.

3. Interprete (python3.3)
    - Ejecute el programa hola.py desde su propio directorio
    - Abra el editor de texto y observe el código del archivo hola.py.

4. Expresiones

El interpretador de Python funciona como una calculadora en la que se escribe una expresión y ella responde con el resultado. La sintaxis de las expresiones sigue la notación matemática tradicional y respeta sus reglas de agrupación. Se puede escribir comentarios que son textos cualesquiera precedidos del carácter #. Al agregar sólo un comentario el intérprete se queda esperando por una acción. (Del manual de Python, capítulo 4) [1].

A continuación se coloca la tabla con los operadores básicos:

Operación	**Sintaxis**
Adición	a + b
División	a / b
División entera	a // b
Potencia	a ** b
Módulo	a % b
Multiplicación	a * b
Negación Aritmética	- a
Negación Lógica	not a
Disyunción Lógica	a or b
Conjunción Lógica	a and b
Substracción	a - b
Comparador Menor	a < b
Comparador Menor o igual	a <= b
Comparador Iguales	a == b
Comparador desigualdad	a != b
Comparador mayor o igual	a >= b
Comparador mayor	a > b

Ejercicios: Ejecute el interpretador de python y realice las siguientes acciones

```
>>> 2+2
4
>>> # Comentarios
... 2+2
4
>>> 2+2 # y un comentario al final de una línea
4
>>> (50-5*6)//4
5
>>> # La división entera retorna el piso:
... 7//3
2
>>> 7//-3
-3
```

Consultar en el manual de Python para más detalle, sección 4.4. [2]

5. Variables y asignación

Al igual que en las calculadoras, las variables representan casillas de memoria en las que se puede almacenar y luego consultar valores. El signo de igualdad (=) es el operador que se utiliza para hacer una asignación a una variable.

Ejercicios: Pruebe en el interpretador de Python las siguientes acciones:

```
>>> width = 20
>>> height = 5*9
>>> width * height
```

Se puede asignar el mismo valor a varias variables al mismo tiempo, colocando el símbolo = entre las variables y valor que se quiere asignar al final.

Ejercicios: Pruebe en el interpretador de Python las siguientes acciones:

```
>>> x = y = z = 0 # Cero para x, y, z
>>> x
>>> y
>>> z
```

Se puede asignar valores a variables en paralelo, de forma similar a como se hace en GCL, así como observar sus valores.

Ejercicios: Pruebe en el interpretador de Python las siguientes acciones:

```
>>> x,y = 1,2
>>> x,y
>>> x,y = y,x
>>> x,y
```

Las variables tienen que estar definidas (lo que en python significa haber sido asignadas) antes de ser utilizadas. De lo contrario ocurre un error.

Ejercicios: Pruebe en el interpretador de Python la siguiente acción:

```
>>> n
…
NameError: name 'n' is not defined
```

6. Valores reales y complejos

Python maneja igualmente valores de punto flotante (representación de un subconjunto finito pero interesante de los reales). En presencia de un operando punto flotante, las operaciones aritméticas convierten los enteros participantes en valores punto flotantes.

Ejercicios: Pruebe en el interpretador de Python las siguientes acciones:

```
>>> 3 * 3.75 / 1.5 # el resultado es 7.5
>>> 7.0 / 2 # el resultado es 3.5
```

Para escribir un número complejo se agrega la letra jota (j) al final del número Los complejos con parte real distinta de cero se escriben (real, imag) o pueden crearse con la operación complex(real, imag).

Ejercicios: Pruebe en el interpretador de Python las siguientes acciones:

```
>>> 1j * 1J   # el resultado es (-1+0j)
>>> 1j * complex(0,1)  # el resultado es (-1+0j)
>>> 3+1j*3    # el resultado es (3+3j)
>>> (3+1j)*3  # el resultado es (9+3j)
>>> (1+2j)/(1+1j)      # el resultado es (1.5+0.5j)
```

Para extraer la parte real o la imaginaria de un complejo se utilizan las operaciones real e imag. Como se muestra a continuación:

```
>>> a=1.5+0.5j
>>> a.real
1.5
>>> a.imag
0.5
```

Para pasar un entero a real se utilizan la operación **float**. Se puede transformar un complejo a real utilizando la operación **abs** que devuelve la norma o módulo de un complejo. Si se utiliza sobre un real devuelve el valor absoluto.

Ejercicios: Pruebe en el interpretador de Python las siguientes acciones:

```
>>> float(3)
>>> abs(-3)
>>> a=3.0+4.0j
>>> float(a)
... # algunas líneas explicativas
TypeError: can't convert complex to float; use abs(z)
>>> a.real # resultado 3.0
>>> a.imag # resultado 4.0
>>> abs(a) # sqrt(a.real**2 + a.imag**2) # resultado 5.0
```

El ambiente interactivo de Python define automáticamente la variable _ (carácter de subrayado) como el valor de la última expresión que se mostró.

Ejercicios: Pruebe en el interpretador de Python las siguientes acciones:

```
>>> impuesto = 12.5 / 100
>>> precio = 100.50
>>> precio * impuesto # el resultado es 12.5625
>>> precio + _        # el resultado es 113.0625
>>> round(_, 2)       # el resultado es 113.06
```

7. Valores Booleanos

Las constantes booleanas se escriben en Python con la primera letra en mayúscula `True` y `False`. Los operadores lógicos en orden de precedencia son

- x or y {y Si x es False, x sino}
- x and y {x si x es False, y sino}
- not x {True si x es False, False sino}

Si se quiere escribir una implicación p⇒ q, se escribe la expresión p<=q. Esto es porque los valores booleanos tienen un orden implícito, donde False < True. Para la equivalencia se utiliza la igualdad (==)

8. Comparadores

Python maneja seis comparadores, todos con la misma prioridad (que es mayor que la de los operadores booleanos). Los comparadores pueden encadenarse arbitrariamente. Por ejemplo, x<y<=z es equivalente a x<y and y<=z. Los comparadores son < (estrictamente menor), <= (menor o igual), > (estrictamente mayor), >= (mayor o igual), == (igual), != (no igual).

9. Cuantificadores acotados y predicados

Python posee cuantificadores sobre rangos acotados que pueden utilizarse para traducir los cuantificadores de la notación GCL. La función range(a,b) retorna la lista de enteros entre a y b (sin incluir b, es decir, a<=x<b). Por ejemplo, al introducir la instrucción rango(10,20), en el interpretador se observa

```
>>> range(10,20)
[10, 11, 12, 13, 14, 15, 16, 17, 18, 19]
```

Para utilizar un predicado debe primero definirse. Por ejemplo, el predicado esPar(x) se puede definir usando el operador módulo (%) , de la siguiente forma

```
>>> def esPar(x) : return x%2==0
…
>>>
```

Un cuantificador acotado en Python consta de tres partes: un rango, un filtro sobre ese rango y un predicado. El rango se puede expresar usando la función **range** y está precedido de la palabra **in**. El filtro es una condición adicional que puede aparecer en los cuantificadores de GCL, dentro del rango; éste se coloca en Python precedido de la palabra reservada **if**. Y el predicado corresponde al cuerpo del cuantificador GCL. La *palabra reservada* usada para el cuantificador universal es **all** y para el cuantificador existencial es **any**. El orden en que se escribe un cuantificador en Python es:

palabraReservada (predicado **`for`** variable **`in`** rango **`if`** filtro)

Por ejemplo, para expresar los valores cuadrados menores que b*10, en GCL se usaría un cuantificador universal (forall x: 0<=x<b: x**2<= b*10). Esta expresión se cumple sólo para algunos valores de b. Para implementar este cuantificador en Python, podemos hacerlo por partes. Primero establecer el rango 0<=x<b, en este caso no hay filtro, y luego el cuerpo del cuantificador usando un predicado para la expresión x**2<= b*10, como se muestra a continuación:

```
all (x**2 <= b*10 for x in range(0,b))
```

Con esta expresión se puede definir un nuevo predicado como sigue:

```
def predicado(b): return all (x**2 <= b*10 for x in range(0,b))
```

Ejercicio: Escriba en el interpretador de Python la definición del predicado y pruebe las siguientes acciones.

```
>>> predicado(19) # el resultado es False
>>> predicado(5)  # el resultado es True
```

Cuando el rango de un cuantificador incluye condiciones adicionales se coloca un filtro. Por ejemplo, el existencial en GCL (exists z: -10<=z<=10 /\ z!=0 : z*z=z+z), incluye la condición adicional z!=0, la cual puede agregarse usando un filtro como se muestra a continuación.

```
def predicado2(): return any (z*z==z+z for z in range(-10,11) if z!=0)
```

Ejercicio: Pruebe en el interpretador de Python la definición del predicado2

También se pueden definir expresiones que usan las funciones de agregación máximo (max), mínimo (**min**) y sumatoria (**sum**), con la misma sintaxis. Por ejemplo, para definir el máximo en el rango 20..30, el mínimo en el rango 20..30 y la sumatoria de los valores en ese rango se usan las expresiones:

```
max (x for x in range(20,31))
min (x for x in range(20,31))
sum (x for x in range(20,31))
```

Ejercicio: Pruebe en el interpretador de Python las expresiones anteriores

10. Precondiciones y postcondiciones

En Python es muy sencillo escribir las precondiciones y postcondiciones de los programas. Esto se hace usando la función *assert* que verifica si una condición se satisface o no. En caso que la condición no se verifique se produce un *AssertionError* y el programa detiene su ejecución. Por ejemplo, la siguiente secuencia de acciones muestra un programa con pre y postcondición.

```
>>>a,b = 1,2
>>>assert(a == 1 and b == 2) #Precondicion
>>>c = a + b
>>>assert(c == a+b) #Postcondicion
```

Ejercicio: Pruebe en el interpretador de Python el programa anterior y observe lo que ocurre al escribir la siguiente aserción

```
>>>assert(a == 1 and b == 1)
```

11. Ejercicios Adicionales

Parte 1: Traducción de variables y expresiones de GCL a Python.
Escriba en el interpretador de Python definiciones equivalente para las siguientes variables de GCL. Como en Python no se colocan los tipo de las variables escriba un comentario al lado de cada una que indique el tipo de la misma.

1. var x := 10:int
2. var edad := 32:int
3. var pi:= 3.14159: float
4. var altura:= 2.1: float
5. var peso:= 76.8: float
6. var y:= 20 + 3i: complejo
7. var z:= 1+i: complejo

Usando las variables previamente definidas, traduzca a Python las siguientes expresiones

1. x mod 3
2. 525 div edad
3. el valor absoluto de -pi
4. la norma de y dividido entre el peso
5. altura por pi

Parte 2: Traduzca a Python las siguientes expresiones en GCL. Luego ejecútelas en el interpretador

1. (sum x : 1 <= x < 100 /\ x mod 7 == 0 : x)
2. (max x : 5 <= x < 40 /\ x*x mod 7== 0 : x)
3. (min x : 5 <= x < 40 /\ x*x == 128 : x)
4. (exists x: 0 <= x <= b : b=2*x)
5. Esperfecto(n) == (exists x: 0 <= x <= n : n=x*x)

Parte 3: Para las siguientes secuencias de instrucciones en Python complete las aserciones o instrucciones indicadas

1.
```
a,b = 32,43
# Escribir Precondicion
b , a = a , b
# Postcondicion
assert( b == 32 and a == 43)
```

2.
```
a,b = 100,5
# Precondicion
assert( a == 100  and b == 5 )
# Agregar una asignación en lugar de este comentario
# Postcondicion
assert( c == a/b )
```

3.
```
a,b = 43, 35
# Precondicion
assert( a == 43 and b == 35 )
# Calculos
c = a % b
c = c*10
a = c
# Escribir Postcondicion
```

4.
```
a,b,c = 80,180,0
# Precondicion
assert( a == 80 and b == 180 )
# Agregar en este punto una secuencia
# tal que cumpla con la postcondición dada.
# sugerencia: Utilizar la función módulo! (%)
#Postcondicion
assert( a == 80 and b == 180 and c == 2000 )
```

Parte 4: Abra el archivo ejemplo1.py en un editor de texto. Cargue el archivo escribiendo el siguiente comando en el interpretador de Python:

>>> from ejemplo1 import *

Usando el mismo modelo de dicho archivo (comentarios, aserciones y estructura) escriba programas en Python que resuelvan los siguientes problemas y guárdelos con los nombres sugeridos:

1. Lab1Ejercicio1.py: Calcular el área de un circunferencia
2. Lab1Ejercicio2.py: Dados tres valores enteros a,b,c que cumplen a>=b>=c, intercambie los valores de manera que cumplan a<=b<=c
3. Lab1Ejercicio3.py: Dice si b es par

Suba los programas en su espacio del aula virtual.

Referencias

[1] The Python Standard Library, Document version 3.3, Disponible en la web: https://docs.python.org/3.3/library/stdtypes.html

[2] The Python Standard Library, Numeric Types: Document version 3.3, Disponible en la web:: https://docs.python.org/3.3/library/stdtypes.html#numeric-types-int-float-complex