Elaboración de funciones y programas

Entrada y salida (datos de entrada y resultado)

Los comandos input y print nos permiten escribir mensajes, ingresar datos (input) y obtener resultados (print).

>>> a = input ("ingrese un numero entero positivo: ")
>>> b = input ("ingrese un numero entero positivo: ")
>>> print "la suma de ", a, " y", b, " es: ", a+b
?	

>>> print "la div entera de ", b , " y", a , " es: ", (b // a, b % a) ?	
>>> print "los numeros entre ", a , " y", b , " son: ", range (a, b+1) ?	

>>> nombre = input ("ingrese su nombre: ")
>>> nombre
?
>>> nombre = input ("ingrese su nombre entre comillas: ")
>>> nombre
?
>>> nombre = raw_input ("ingrese su nombre: ")
>>> nombre
?	

>>> z = input ("ingrese un numero positivo: ")
>>> z
?
>>> z = raw_input ("ingrese un numero positivo: ")
>>> z
?	

Input y raw_input

Si consideramos por ejemplo el símbolo 3, ¿qué es? O como preguntamos en informática: ¿qué tipo tiene? Observar que el 3 es una constante que puede ser un dígito que es lo que representa el ***carácter*** 3 (por ejemplo en el teclado, o puede ser un ***número entero***.

Por otro lado, la palabra hola, o la letra a, pueden ser ***nombres de variables*** o pueden ser ***strings (secuencias de caracteres).*** En general, en el caso de strings, en un lenguaje de programación hay que ponerlos entre comillas. Para facilitar la entrada de datos, python proveee esas dos funciones, donde usamos **input** para ingresar datos de tipo numérico y **raw_input** para ingresar caracteres o strings, sin necesidad de indicarlos entre comillas. Sin embargo, Python no es un lenguaje con control de tipos, como otros, por ejemplo Pascal, Haskell, etc. Observar con el siguiente ejemplo:

```
>>> z = input ("ingrese un numero positivo: ")
ingrese un numero positivo: 'a'
>>> z
'a'
```

que el lenguaje Python tiene “errores y/o ambigüedades” y que el programador ***siempre*** debe tener el control de datos, variables e instrucciones.

Selección

Las instrucciones de selección permite implementar acciones como “si llueve me quedo en casa, si no, voy a clase”. En Python tenemos dos instrucciones de selección:

```
if (condición):
    instrucción
    …
if (condición):
    instrucción
    …
else:
   instrucción
   ...
```

Funciones

En los ejercicios arriba, hemos ingresado datos (por ejemplo, los números a y b), hemos operado con ellos y hemos obtenido un resultado (usando el comando print). Las operaciones que hemos realizado corresponden a *funciones predefinidas* en Python (+, //, %, range). Podemos también definir nuestras propias funciones para operar con los datos de entrada y solucionar diferentes problemas.

Ejemplo: supongamos que dado un número enteros positivo, queremos obtener la lista de sus divisores.

Por ejemplo, si el número es 86, queremos obtener la lista [1, 2, 43, 86], que son sus divisores. No existe en Python una función predefinida que podamos usar para ello. Tenemos que definirla nosotros.

Supongamos que la hemos definido y se llama “divisores”, podemos escribir entonces un programa así:

print "bienvenido al programa que calcula los divisores de numeros enteros positivos”

a = input ("ingrese un numero entero positivo: ")

print "los divisores de ", a, "son ", divisores(a)

A la hora de imprimir el resultado, Python aplica la función divisores que hemos definido al número que hemos ingresado y nos da la lista. Podemos ingresar 86 y tendremos [1, 2, 43, 86]. Si queremos la lista para otro número, volvemos a ejecutar el programa. Lo ejecutamos repetidamente para todos los números que queramos … o iteramos!!!

Iteraciones

Una iteración es una instrucción que nos permite repetir una secuencia de instrucciones, es decir, es una manera de decirle al computador, haga esto para todos los datos de entrada que se ingresen, sin tener que volver a ejecutar el programa cada vez. Veamos el ejemplo de la función divisores:

```
0. # divisores :: N - {0}→ [N]
1. def divisores (a):
2.     lista = [a,1]
3.     i = 2
4.     while (i < a):
5.         if (a % i == 0): lista = lista + [i]
6.         i = i + 1
7. return lista
```

línea 0: como Python no tiene control de tipos, debemos incluir un comentario que indique que los datos de entrada deben ser números enteros (Z es la denominación matemática del conjunto de números enteros). Esta es una información muy útil por ejemplo para re-usar una función o encontrar errores.

En la línea 1. se indica que lo que se va a definir es una función que llamamos “divisores”, tiene un solo dato de entrada, que indicamos con la variable a.

Explicaremos las líneas de la 2 a la 6. Recordemos los ejemplos de algoritmos que vimos anteriormente (slides 3 y 5 del primer encuentro). En la descripción del algoritmo para ambos problemas (slide 6), se dice “dada una lista de elementos, obtener un resultado operando con sus elementos, a partir de un elemento inicial.”

En la línea 2, **lista** representa el resultado que se construirá a partir del valor inicial **[1,a]**. (¿Por qué ese valor?).

En la línea 5 se opera con a y con la lista, de la siguiente manera: se agrega a la lista un valor **i**, que es divisor de a (**a % i == 0** es verdadero si el resto de dividir a por i es 0, por lo tanto i es divisor de a).

En la línea 3 se indica el valor inicial de la variable **i** que representa a los divisores (¿por qué es 2?) y en la línea 6 se pasa al siguiente valor eventualmente divisor de a.

En la línea 4, indicamos que se repita la instrucción de la línea 5 y 6 para 2, 2+1, 2+1+1, 2+1+1+1, … hasta llegar a a-1.

Sintaxis de la instrucción de iteración **while**

Sea i una variable enumerable (de tipo entero, carácter), se define la instrucción while así:

```
i = valor inicial
while (condicion(i)):
    instruccion
    instruccion
    …
    i = nuevo valor
```

Semánticamente significa que se ejecutan los siguientes pasos:

1. se evalúa la condición dependiendo del valor inicial de i
2. si es verdadera, se ejecutan las instrucciones dentro del while, que incluyen asignar a i un nuevo valor
3. se evalúa la condición con el nuevo valor de i y se repite el paso 2
4. si la condición resulta false, se termina la iteración

Atención: las instrucciones dentro del while del paso 2 quedan determinadas por la ***indentación,*** es decir, por los espacios en blanco que las alinea. Por ejemplo:

```
i = 0
while (i<10):
   print 1
i = i+1
```

produce un ***loop infinito*** dado que el valor de i nunca se actualiza dentro del while. Esto significa que la condición **i<10** es siempre verdadera pues se evalúa siempre para **i = 0**.

Un ejemplo usando raw_input

Podemos usar una condición que dependa de si el ususario quiere continuar o no.

El siguiente programa ingresa números enteros de la entrada estándar e imprime los que son pares

```
# espar : Z → {True, False}
def espar(x):
    return x%2 == 0
# programa principal
continuar = 's'              # suponemos que siempre se ingresa al menos un numero
while (continuar == 's'):
     numero = input ("ingrese un numero entero: ")
     if espar(numero):
        print "el numero ", numero, " es par"
     continuar = raw_input ("ingrese s para continuar o n para detener: ")
```

Observar que la instrucción print "el numero ", numero, " es par" **está indentada** más adentro que el resto. Esto es porque de esa forma Python puede reconocer que lo que debe ejecutarse si la condición del **if** es verdadera es lo que está indentado más adentro.

Iteraciones

En muchos de los problemas que solucionamos o de las tareas que resolvemos aplicamos métodos que se basan en la repetición de acciones. Para programar esos métodos (es decir, para poder enseñarlos a un computador), contamos con instrucciones de ***iteración*** o de ***recursión***.

Las instrucciones de iteración utilizan una variable de control de la iteración, para establecer dónde empieza y dónde termina la iteración, mientras que las instrucciones de recursión se basan en hacer lo mismo sobre algo que va disminuyendo (lo veremos más adelante).

La instrucción while que hemos usado en los ejemplos podría sustituirse con una instrucción for. Por ejemplo, los algoritmos de las slides se pueden implementar con for:

```
def listaCI(lista):
     resultado = []
     largo = len(lista)        # observar que no restamos 1
    for i in range(largo):
          resultado = resultado + [lista[i][1]]
 return resultado
```

donde la variable i se inicializa y actualiza dentro del for.

En algunos casos, aunque el resultado puede ser el mismo, desde el punto de la programación NO es lo mismo y se considera error usar la isntrucción no adecuada. Por ejemplo, si buscamos un elemento en una lista no es lo mismo usar una que otra … ¿Por qué?

Ejercicios

Sea el ejemplo usando raw_input.

Implementar la función “espar” que dado un número entero positivo determina si es par o no. Implementar un programa principal que lea números enteros positivos ingresados de la entrada estándar (el teclado), determine para cada uno de ellos si es par o no e imprima el mensaje del ejemplo.

Para usar el compilador

Para escribir un programa se debe usar un editor de texto (gedit (recomendado), block de notas, etc) , **editar el programa** siguiendo estrictamente las reglas de sintaxis que hemos visto (indentación, **:** en algunas instrucciones como def, if, while), etc, y guardarlo en **la misma carpeta en la que instalaron Python.**

Una vez hecho esto **ejecutan** el programa con el comando **python nombre.py** (suponemos que el programa lo guardaron con el nombre nombre.py). Es necesario poner **.py** a continuación del nombre que han elegido. Si no lo hacen, o si lo guardan en otra carpeta, obtendrán un error:

python: can't open file 'nombre': [Errno 2] No such file or directory

El programa se ejecuta entonces con **python nombre.py**. Si consta de instrucciones “input” o “raw_input” deben ingresar los datos. Por ejemplo, para este programa, que escribimos en un archivo que denominamos **progslides.py**:

```
# programa para el algoritmo de la suma de salarios (slide 3).

# listaCI : lista ((nombre, CI, emails)) -> lista(CI)

def listaCI(lista):
    resultado = []
    i = 0
    largo = len(lista)-1
    while (i <= largo):
        resultado = resultado + [lista[i][1]]
        i = i + 1
    return resultado

# programa principal para sumasalarios

lista = input ("ingrese una lista de (nombre, CI, salario): ")

print "la suma de los salarios es ", sumasalarios(lista)
```

tenemos la siguiente ejecución

$ python progslides.py

ingerese una lista de (nombre, CI, salario): [('sylvia', 2222, 345.0),('ana',3333,455.5),('pedro',8888, 298.6),('juan',7777,948.0)]

la suma de los salarios es 2047.1

donde la lista **[('sylvia', 2222, 345.0),('ana',3333,455.5),('pedro',8888, 298.6),('juan',7777,948.0)]**

la hemos ingresado desde el teclado (observar que los nombres hay que ponerlos entre comillas). Es una lista de ternas (string, número, número), como dice la especificación del problema.

Una utilidad extra de los comentarios

Podemos escribir varios programas en un mismo archivo y a la hora de ejecutar, vamos comentando los que no ejecutamos. Por ejemplo, escribo los dos programas de las slides 3 y 5 en el mismo archivo **progslides.py,** ejecuto el primero comentando el segundo y luego ejecuto el segundo comentando el primero. (Recordar que los comentarios son ignorados por el compilador). Para comentar más de una línea se usan las ''' **.** La parte de texto que queda entre ''' no se ejecuta. Cuando uno de los programas esté trabajado, se comenta el otro. De esa forma evitamos escribir varios archivos.

```
# programa para el algoritmo slide 3

# sumasalarios : lista ((nombre, CI, salario)) -> Z

def sumasalarios(lista):
  suma = 0
  i = 0
  largo = len(lista)-1
  while (i <= largo):
     suma = suma + lista[i][2]
     i = i + 1
  return suma
'''                                          # aca comienza lo que no se ejecutará
# programa para el algoritmo slide 5

# listaCI : lista ((nombre, CI, emails)) -> lista(CI)

def listaCI(lista):
  resultado = []
  i = 0
  largo = len(lista)-1
  while (i <= largo):
     resultado = resultado + [lista[i][1]]
     i = i + 1
  return resultado
'''                                          # aca termina
```

```
# programa principal para sumasalarios
lista = input ("ingrese una lista de (nombre, CI, salario): ")
print "la suma de los salarios es ", sumasalarios(lista)
'''                                  # sigue texto que será ignorado
# programa principal para listaCI
lista = input ("ingerese una lista de (nombre, CI, email): ")
print "la lista de las CI es ", listaCI(lista)
'''                                  # fin de texto ignorado
```

Ejercicios

1. Dados **programa1, programa2 y programa3** edítelos y guárdelos en su carpeta donde tiene instalado Python. Anticipe que debería ir en los "**...**" del primer mensaje. Luego, ejecute los programas (con python … según se explicó arriba) y verifique sus respuestas. Reflexione sobre el uso de la indentación y de las sentencias **if** (con y sin **else**).
2. Abra y lea el archivo **programa4.py** (para abrir un archivo.py utilice el editor de texto). Responda el comentario del principio (para ello deberá guardarlo (en su carpeta python) y ejecutarlo). Reflexione sobre los distintos resultados según la indentación.
3. Escriba programas usando la sentencia (o instrucción) **for** para los algoritmos de las slides 3 y 5.