Conceptos Básicos

1.- ¿Qué quiere decir el hecho de que Python es un lenguaje interpretado?

¿Cuáles son sus ventajas y desventajas? Compare con Pascal o con otro

lenguaje que haya utilizado.

**Se ejecuta utilizando un programa intermedio llamado intérprete, en lugar de compilar el código a lenguaje máquina que pueda comprender y ejecutar directa-mente una computadora (lenguajes compilados)**

**Ventajas  
\* más flexibles y más portables.  
\* Desarrollo más rápido : Puedes escribir un programa, salvarlo y ejecutarlo. En un lenguaje compilado tienes que pasar por los pasos de compilar y ligar el software, lo cual puede ser un proceso lento.  
\* Multiplataforma : El mismo código funciona en cualquier arquitectura, la única condición es que disponga del intérprete del lenguaje. No es necesario compilar el código una vez para cada arquitectura.**

**Desventajas  
\* Lentitud : Los programas interpretados son más lentos que los compilados. Sin embargo los programas interpretados suelen ser cortos, en los que la diferencia es inapreciable.**

2.- ¿A qué se refiere con que el lenguaje sea multiplataforma? ¿Cómo es esto posible en Python?

**El intérprete de Python está disponible en multitud de plataformas (UNIX, Solaris, Linux, DOS, Windows, OS/2, Mac OS, etc.) por lo que si no utilizamos librerías específicas de cada plataforma nuestro programa podrá correr en todos estos sistemas sin grandes cambios.**

3.- ¿Por qué es software libre? ¿Qué ventajas ven en esta característica?

**«Software libre» significa que el software respeta la libertad de los usuarios y la comunidad. En términos generales, los usuarios tienen la libertad de copiar, distribuir, estudiar, modificar y mejorar el software. Con estas libertades, los usuarios (tanto individualmente como en forma colectiva) controlan el programa y lo que hace.**

**Ventajas:**

**La libertad de usar el programa, con cualquier propósito (libertad 0).**

**• La libertad de estudiar cómo funciona el programa, y adaptarlo a necesidades propias (libertad 1).**

**•El acceso al código fuente es una condición previa para esto.**

**• La libertad de distribuir copias (libertad 2).**

**• La libertad de mejorar el programa y hacer públicas las mejoras a los demás, de modo que toda la comunidad se beneficie. (libertad 3).**

**•El acceso al código fuente es un requisito previo para esto**

4.- Python es de tipado dinámico. ¿Qué significa eso?

**La característica de tipado dinámico se refiere a que no es necesario declarar el tipo de dato que va a contener una determinada variable, sino que su tipo se determinará en tiempo de ejecución según el tipo del valor al que se asigne, y el tipo de esta variable puede cambiar si se le asigna un valor de otro tipo.**

Primero pasos ...

5.- Ejecute las siguientes instrucciones:

print "Hola Mundo"

print "Mi primer script"

Ahora modifíquelo de la siguiente manera:

print "Hola Mundo Mi primer script"

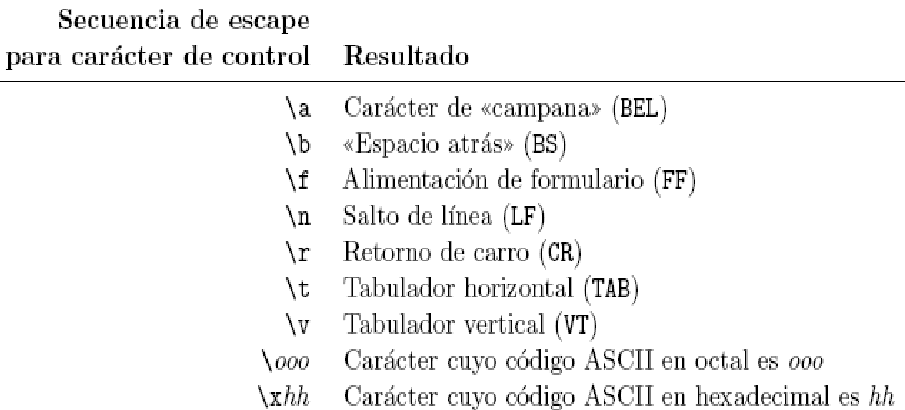
a) ¿Qué hace falta para que esa misma línea tenga el mismo efecto que la anterior?

b) ¿Qué otros caracteres de control conoce? Muestre un ejemplo en el que la salida sea algo similar a: (en una única sentencia)

Uno Dos Tres

Cuatro Cinco Seis

\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*



6- a) Escriba un script que almacene dos números en dos variables eimprima el resultado de la suma de ambos.

b) ¿Qué ocurre si asigna los números a las variables con“”, por ejemplo num1 = “3”? ¿Qué devuelve la suma?

c) ¿Qué sucede si en vez de la suma queremos hacer la multiplicación en a) y b)?

d) ¿Qué resultado devuelve si al final de la sentencia de impresión de a) y b) agregamos \* 5?

7.- Escriba un script que reciba desde la entrada de datos estándar (teclado) una letra o número y lo devuelva de la siguiente manera:

print "El caracter ingresado es: "+ caracter

Ayuda: investigue el comando raw\_input()

En algún momento nuestra aplicación tendrá que preguntar al usuario que desea hacer, o algun dato necesario que utilizará posteriormente o también elegir entre varias opciones. Sea cual sea el caso necesitamos que el programa interactúe con él y le pregunte. Para ello hay una función específica en python que nos ayudará en esta tarea. La función es raw\_input() y guarda lo que se introduzca en  
formato string y se llamaría asi:

|  |  |
| --- | --- |
| 1 | raw\_input("Frase si se quiere ") |

Se puede colocar una frase para que el usuario sepa que ha de introducir sino se introduce frase se puede colocar:

|  |  |
| --- | --- |
| 1 | raw\_input() |

De la forma que lo colocamos es correcta pero de poco nos sirve… ya que lo que el usuario haya introducido el programa no lo recordará, debemos asignarlo a una variable como se realiza:

|  |  |
| --- | --- |
| 1 | recuerdo = raw\_input("Introduzca lo que quiere que recuerde: ") |
|  |  |

8.- a) Modifique el ejercicio 6) para que los valores a ser sumados sean datos ingresados por el usuario, indicando que debe ingresar un número por pantalla y luego otro.

b) ¿El resultado fue el esperado? ¿Qué debería modificar para que dé lo esperado (suma de números)?

9.- a) Implemente una calculadora simple, en donde se ingrese (por entrada estándar) dos operandos y el operador(+,-,\*, /) e imprima el valor de la operación resultante(por el momento no tenga en cuenta errores de tipos, ej: que el operando no sea número o que el operador no sea los enumerados).

Nota: No codifique con condicionales (if, elif, etc) ni bucles (while, for, etc)

Ayuda: Investigue el comando eval()

Ésta es la última pieza del rompecabezas (o mejor dicho, del solucionador de rompecabezas). Tras toda esa sofisticada manipulación de cadenas, lo que nos queda es una cadena parecida a '9567 + 1085 == 10652'. Pero eso es una cadena. ¿De qué nos sirve una cadena? Te presento a eval(), la herramienta universal de evaluación de Python.

>>> eval('1 + 1 == 2')

True

>>> eval('1 + 1 == 3')

False

>>> eval('9567 + 1085 == 10652')

True

Pero espera, ¡hay más! La función eval() no se limita a expresiones booleanas. Puede manejar *cualquier* expresión de Python y devuelve *cualquier* tipo de dato.

>>> eval('"A" + "B"')

'AB'

>>> eval('"MARK".translate({65: 79})')

'MORK'

>>> eval('"AAAAA".count("A")')

5

>>> eval('["\*"] \* 5')

['\*', '\*', '\*', '\*', '\*']

Pero espera, ¡que eso no es todo!

>>> x = 5

>>> eval("x \* 5") ①

25

>>> eval("pow(x, 2)") ②

25

>>> import math

>>> eval("math.sqrt(x)") ③

2.2360679774997898

1- La expresión que toma eval() puede hacer referencia a variables globales definidas fuera de eval(). Si se invocase dentro de una función también puede hacer referencia a variables locales.

2- Y a funciones.

3- Y módulos

Eh, espera un momento…

>>> import subprocess

>>> eval("subprocess.getoutput('ls ~')") ①

'Desktop         Library         Pictures \  
 Documents       Movies          Public   \  
 Music           Sites'

>>> eval("subprocess.getoutput('rm /un/fichero/cualquiera')") ②

|  |  |
| --- | --- |
| ① | El módulo subprocess te permite ejecutar órdenes arbitrarias en el sistema y obtener su resultado como una cadena de Python. |
| ② | Las órdenes arbitrarias pueden tener consecuencias permanentes. |

Peor aún, porque existe una función global \_\_import\_\_() que toma una cadena con el nombre de un módulo, lo importa y devuelve una referencia a él. Combinado con la potencia de eval(), puedes construir una expresión que destruya todo tus ficheros:

>>> eval("\_\_import\_\_('subprocess').getoutput('rm /un/fichero/cualquiera')") ①

|  |  |
| --- | --- |
| ① | Ahora imagina la salida de 'rm -rf ~'. En realidad no habría salida alguna, al igual que tampoco te quedarían ficheros. |

**eval() is EVIL**

Es decir, eval() es maligno. Bueno, es maligno si se evalúa expresiones arbitrarias provenientes de fuentes no contrastadas. Sólo deberías usar eval() sobre entradas confiables. Por supuesto, el truco está en averiguar qué es lo “fiable”. Pero hay algo de lo que estoy seguro: **NUNCA** deberías tomar este solucionador de alfaméticas y colgarlo en internet a modo de pequeño y divertido servicio web. No cometas el error de pensar “Diablos, la función hace gran cantidad de manipulaciones antes de obtener una cadena que evaluar; *no puedo imaginar* cómo podría alguien explotar esto”. Alguien **AVERIGUARÁ** cómo colar malvado código ejetutable a través de toda esa manipulación de cadenas ([cosas más raras han pasado](http://www.securityfocus.com/blogs/746)) y ya puedes despedirte de tu servidor.

Pero seguramente debe haber *alguna* manera de evaluar expresiones de forma segura, ¿no? De poner eval() en un entorno seguro donde no pueda acceder o dañar al mundo exterior. Bueno, sí y no.

>>> x = 5

>>> eval("x \* 5", {}, {}) ①

Traceback (most recent call last):

File "<stdin>", line 1, in <module>

File "<string>", line 1, in <module>

NameError: name 'x' is not defined

>>> eval("x \* 5", {"x": x}, {}) ②

>>> import math

>>> eval("math.sqrt(x)", {"x": x}, {}) ③

Traceback (most recent call last):

File "<stdin>", line 1, in <module>

File "<string>", line 1, in <module>

ameError: name 'math' is not defined

1-El segundo y el tercer parámetros pasados a la función eval() actúan como los espacios de nombres global y local durante la evaluación de expresiones. En este caso, ambos están vacíos, lo que significa que cuando se evalúa la cadena "x \* 5", no existe una referencia a x en el espacio de nombres global ni en el local, de manera que eval() lanza una excepción.

2- Puedes incluir de forma selectiva valores específicos en el espacio de nombres global listándolos individualmente. Entonces esas — y sólo esas — variables estarán disponibles durante la evaluación.

3-Incluso aunque acabas de importar el módulo math, no lo has incluido en el espacio de nombres que pasaste a eval(), así que la evaluación falla.

Vaaaaya, eso fue fácil. ¡Déjame hacer mi servicio web alfamético ahora!

>>> eval("pow(5, 2)", {}, {}) ①

25

>>> eval("\_\_import\_\_('math').sqrt(5)", {}, {}) ②

2.2360679774997898

1- Incluse aunque pases diccionarios vacíos como espacios de nombres global y local, todas las funciones integradas de Python siguen estando disponibles durante la evaluación. Por tanto, pow(5, 2) funciona, porque 5 y 2 son literales, y pow() es una función integrada.

2-Desafortunadamente (y si no ves por qué es desafortunado, sigue leyendo), la función \_\_import\_\_() se encuentra entre las integradas, así que esto también funciona.

Sí, eso significa que sigues pudiendo hacer cosas feas, incluso si indicas de forma explícita que los espacios de nombre global y local van a ser diccionarios vacíos cuando invocas eval():

>>> eval("\_\_import\_\_('subprocess').getoutput('rm /un/fichero/cualquiera')", {}, {})

Ups. Me alegro de no haber hecho ese servicio web alfamético. ¿Hay *alguna* manera de usar eval() de forma segura? Pues sí y no.

>>> eval("\_\_import\_\_('math').sqrt(5)",

...     {"\_\_builtins\_\_":None}, {}) ①

Traceback (most recent call last):

File "<stdin>", line 1, in <module>

File "<string>", line 1, in <module>

NameError: name '\_\_import\_\_' is not defined

>>> eval("\_\_import\_\_('subprocess').getoutput('rm -rf /')",

...     {"\_\_builtins\_\_":None}, {}) ②

Traceback (most recent call last):

File "", line 1, in <module>

File "", line 1, in <module>

NameError: name '\_\_import\_\_' is not defined

1-Para evaluar de forma segura expresiones de origen no fiable, hay que definir un diccionario para el espacio global que asigne el valor nulo de Python (None) a"\_\_builtins\_\_". Internamente, las funciones “integradas” (*built-in*) están contenidas en un pseudomódulo llamado "\_\_builtins\_\_". Este pseudomódulo (es decir, el conjunto de funciones integradas) queda disponible a las expresiones evaluadas a menos que lo quites de en medio explícitamente.

2-Asegúrate de haber sustituido \_\_builtins\_\_. No \_\_builtin\_\_, \_\_built-ins\_\_ o alguna otra variante que funcionará igual de bien pero te dejará expuesto a riesgos catastróficos.

¿Así que ahora eval() es seguro? Bueno, sí y no.

>>> eval("2 \*\* 2147483647",

...     {"\_\_builtins\_\_":None}, {}) ①

1. Incluso sin acceso a \_\_builtins\_\_, se puede seguir lanzando un ataque de negación de servicio. Por ejemplo, intentar elevar 2 a 2147483647 hará subir el uso de CPU de tu servidor al cpu durante un buen rato (si has probado esto en el intérprete interactivo, pulsa Ctrl-C unas cuántas veces para detenerlo). Técnicamente, esta expresión *devolverá* un valor en algún momento, pero mientras tanto tu servidor estará haciendo un montón de… nada.

En el fondo, *es* posible evaluar expresiones arbitrarias en Python de forma segura, para cierta definición de “seguro” que resulta no ser terriblemente útil en la vida real. Está bien si lo que estás haciendo es jugar o si sólo vas a pasarle entradas completamente confiables. Pero cualquier otra cosa es buscar problemas.

b) A la calculadora implementada en a) agréguele la capacidad de calcular la potencia, parte entera y resto de la división. ¿Tuvo que modificar el código realizado en a)?

c) Modifique la calculadora para que reciba dos números enteros y un operando, que pueden ser&, | o^ y corresponden a AND, OR y XOR respectivamente, los transforme en números binarios y calcule la operación ingresada y devuelva el resultado del cómputo tanto en decimal como en binario.

10.- a) Escriba un script que reciba un carácter por entrada estándar e imprima su correspondiente código ASCII en pantalla.

b) Haga la inversa de a), reciba un número y devuelva a que carácter corresponde ese código ASCII

11.- Salida estándar (print)

a) Explique que hace la siguiente línea de código

print "%15s %s Python"% ("Seminario","de")

Lo que hace la que realiza la siguiente línea es convertir en string por medio de %s y luego

b) Suponga que tengo la siguiente variable

numeroFlotante= 7.55689245

Y quiero que se impriman sólo el número con5 dígitos, como debería formular la instrucción print?

c) Dados el nombre de un alumno y su promedio, investigue cómo realizar la impresión de los datos en pantalla sin tener que convertir el valor numérico a string.

Es decir, sin imprimirlo de la forma:

print nombreAlumno+ str(promedio) ni print nombreAlumno, promedio

d) Cómo podría lograr con la instrucción print la siguiente salida:

Fernandez, Gaston 5.56

Sin completar con blancos manualmente luego del nombre y con la información de la siguiente manera("Fernandez","Gaston",5.56)

**Listas , Tuplas y Diccionarios**

12.- Listas

Dada la siguiente lista

lista= ['elemento1', 2, 'elemento3', 'elemento4']

a) Imprima el primer y último elemento

b) Que retorna lista[-1]?

Retorna el elemento 4

c) Imprima todos los elementos

d) Imprima los elementos 0 , 1 y 2

e) Imprima ahora los últimos 3 elementos (sin saber a priori cuantos tiene la

lista)

f) En la segunda posición ingrese el valor “elemento 2”

g) ¿Qué sucede si ejecutamos el siguiente código? ¿Por qué? ¿Qué debería

hacer para agregar un nuevo elemento al final de la lista?

lista[4]='elemento5'

Lo que sucede es que el elemento 2 se agrega en la posición 2 de la lista provocando pisar el 2 y para agregar un nuevo elemento al final de la lista disponemos de una función ***append,*** añade un único elemento al final de la lista.

h) Elimine el elemento creado en g)

i) ¿Cómo puedo hacer para retornar el valor del último elemento y eliminarlo de la lista en una sola instrucción?

Se puede hacer utilizando la funcion pop, que nos permite borrar el último elemento de la Lista como si de una cola se tratara. Esta función nos devuelve el último elemento de la lista y lo borra de ella.

j) Retorne el índice que tiene el elemento con el valor: “elemento 4”

k) Ordene en forma inversa la lista e imprima todos los elementos

l) Cree un string que contenga al menos cinco palabras, sepárelas y ordénelas alfabéticamente y en orden inverso. Imprima cada resultado y la cantidad de palabras del string.

13.- Tuplas

a. ¿Qué son las tuplas? ¿Cómo se definen? ¿Cuál es la diferencia principal con las listas?

Son colecciones de datos ordenados y se definen de la siguiente manera:

tupla1=1,2 o tupla1=(1,2)

–Tupla de un elemento: tupla1=(1,) , caso contrario

tupla1=(1) sería un entero.

–Tupla de ningún elemento: tupla1=()

Y las diferencias principales son:

- Que las listas usan corchetes [] y las tuplas ()

-Son INMUTABLES, su tamaño y los valores de las mismas NO pueden cambiar una vez creada

-Son mas ligeras que las listas en la hora del uso de memoria

- las tuplas pueden ser heterogéneas (es decir, sus entradas tienen significados diferentes), mientras que las listas son homogéneas. Eso es, puedo tener una tupla (int, String, String), pero una lista puede contener sólo un tipo de datos

b. Sea la siguiente tupla:

tupla= (1,[3,4,5])

I. Diga cómo está compuesta la tupla.

Esta compuesta por un numero entero y una lista de numero enteros.

II. ¿Cuántos elementos tiene? Demuéstrelo mediante un script

III. ¿Se pueden cambiar los valores de algún elemento? ¿Se puede agregar un elemento nuevo a la lista?

Se puede cambiar ya q esta conformada por una lista, es decir q se puede cambiar el elemento dos y agregar en la misma lista

IV. ¿Para qué son especialmente útiles las tuplas?

* Las tuplas son más rápidas que las listas. Si define un conjunto constante de valores y todo lo que va a hacer es iterar sobre ellos, use una tupla en lugar de una lista.
* "Proteger contra escritura" los datos que no necesita cambiar hace el código más seguro. Usar una tupla en lugar de una lista es como tener una sentencia *assert*implícita mostrando que estos datos son constantes, y que se necesita una acción consciente (y una función específica) para cambiar eso.
* ¿Recuerda que dije que las [claves de diccionario](http://es.wikibooks.org/wiki/Inmersi%C3%B3n_en_Python/Tipos_de_datos_nativos/Presentaci%C3%B3n_de_los_diccionarios#Modificar_diccionarios) pueden ser enteros, cadenas y "unos pocos otros tipos"? Las tuplas son uno de estos tipos. Se pueden usar como claves en un diccionario, pero las listas no. En realidad, es más complicado que esto. Las claves de diccionario deben ser inmutables. Las tuplas son inmutables, pero si tiene una tupla de listas, eso cuenta como mutable, y no es seguro usarla como clave de un diccionario. Sólo las tuplas de cadenas, números y otras tuplas seguras para los diccionarios pueden usarse como claves.

14- Conjuntos

a. ¿Qué son los sets en Python?

Un conjunto es una colección no ordenada de elementos distintos

* Desordenada: los artículos se levantó por valor, en lugar de ubicación
* Distinct: cualquier valor aparece como máximo una vez

Un conjunto es una colección de datos de igual o diferente tipo, desordenada y sin elementos duplicados.

b. ¿Qué tipo de operaciones se pueden hacer sobre ellos?

– Saber si un elemento pertenece o no al conjunto

– Operaciones matemáticas como la unión, intersección, diferencia y diferencia simétrica, in.

c. ¿Qué particularidad tienen estas estructuras de datos?

d. Sea el siguiente código:

conjunto= set([1,2,3,4,5,6])

i. Agregue el elemento 7 e imprima los elementos del conjunto

ii. Agregue el elemento 4 e imprima los elementos del conjunto. ¿Hubo

alguna modificación? ¿Por qué?

No hubo modificación ya que no puede haber elementos duplicados

iii. Realice la intersección, unión y diferencia del conjunto dado con este otro:

set([3,4,5,10,15])

15.- Diccionarios

a. ¿Qué son los diccionarios? ¿Qué diferencias tienen con las estructuras vistas anteriormente?

Los Diccionarios tambien llamados matrices asociativas , ya que son colecciones que contiene clave:valor

En la parte de la clave podemos usar cualquier tipo de valor inmutable: numeros, candenas, booleanos o [tuplas](http://pycol.blogspot.com/2009/02/tuplas-en-python_27.html) menos [listas](http://pycol.blogspot.com/2009/02/tuplas-en-python.html) o diccionarios dado que son mutables. La difrencia que existe entre diccionarios con la listas o las tuplas es que los valores almacenados en el diccionario no se le accede a traves de su indice si no atraves de su clave utilizando el operador de [] asi que de esta forma nuestros diccionarios no tienen un orden prestablecido :|

b. Sea el siguiente diccionario

personas\_edades={25:'Juan',33:'Florencia',17:'Pedro',40:'Alberto'}

En donde la clave identifica a la edad y el valor al nombre

i. Imprima la cantidad de elementos del diccionario

ii. Imprima el nombre de la persona que tenga 33 años

iii. Imprima todas las edades de las personas del diccionario

iv. Imprimir todas las personas del diccionario

v. Imprimir si una determinada edad está en el diccionario

vi. Modifique el nombre de Juan por Matías

vii. Agregue a Pedro con una edad de 44 años

viii. Elimine el elemento ingresado en el punto anterior

ix. Elimine todos los elementos del diccionario