iLLiCe's Site

Investigación y Desarrollo

Esto es un proyecto llevado a cabo por:
iLLiCe
http://www.illice.net

Plataforma utilizada:
Intérprete Python
Programación Python

Índice

Introducción.....3

Tipos de Datos.....3

Relaciones de Expresiones.....3

Entrada / Salida.....4

Estructuras.....5

Programación modular (Funciones).....7

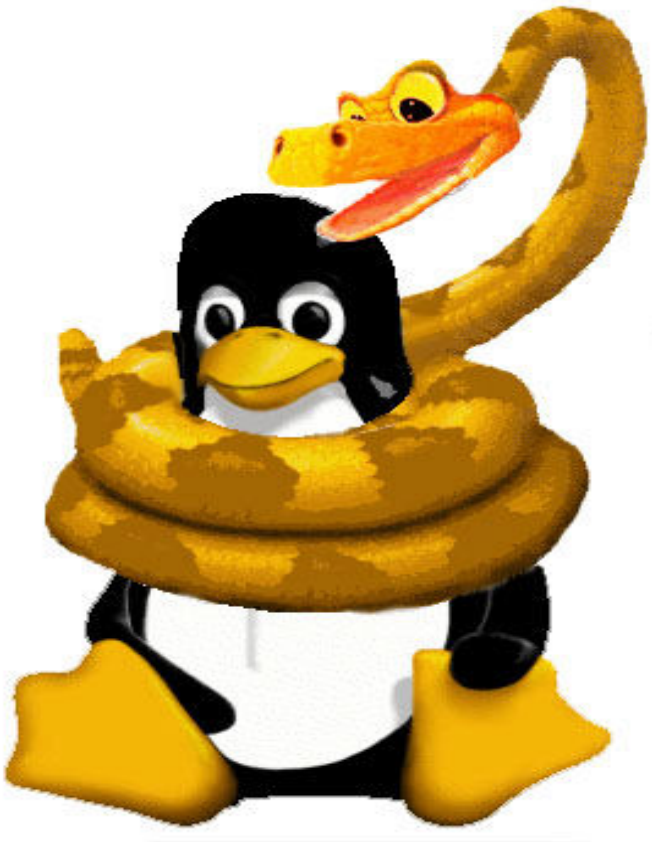
Tablas o Listas.....8

Registros.....9

Ficheros.....9

Documentación oficial sobre Python.....10

Agradecimientos.....10



Introducción

Este manual se ha creado para aquél que no sepa nada de PYTHON ni programación en general, la explicación que aquí doy es muy básica y breve, por lo que sí ya sabes otros lenguajes como PHP, C/C++ ... vale la pena que te detengas y no sigas leyendo ya que esto es para las personas que se están iniciando en la programación.

Tipos de datos

En un programa los datos pueden estar almacenados en variables o constantes. Una constante es un dato que no va a variar durante la ejecución del programa. Una variable si que puede variar su contenido durante la ejecución.

Existen varios tipos de datos, los más sencillos son los enteros, flotantes y de caracteres. Las variables de tipo entero almacenan datos de números enteros, los decimales son los flotantes y los alfanuméricos son los del tipo carácter.

```
Entero →-3,-2..2,3...
Flotante →1.26,0.4,...
Caracteres → a,t,5,x...

Entero → int
Flotante →  float
Caracter → char
```

En la parte superior los tipos de datos simples.

En la parte inferior las equivalencias en lenguajes de programación.

Las variables dependiendo del lenguaje de programación pueden declararse de una forma u otra.

Para declarar una variable en Python no es necesario expresar del tipo que es, es decir, si tenemos la variable a=3, Python reconoce el tipo de dato en este caso entero, pero si ponemos a="Texto" Python ya sabe que esa variable contiene caracteres y trabajará con ella sabiendo es una variable de carácter. Y si ponemos a=3.0, ¿será entero el tipo? Pues no, al poner un .x aunque sea un .0 Python lo interpreta como un flotante.

Relaciones entre expresiones

```
Suma → +
Resta → -
Multiplicación → *
División → /
Resto → %
Potencia → ^ ó **
```

Estas son las expresiones que más se utilizan en los lenguajes de programación, donde todas son muy similares. También existen otros operandos que son los llamados lógicos:

```
Y → and
O → or
no → not
```

Los operadores lógicos se usan para comparar expresiones, o para declarar condiciones, y se usan igual que en la lógica. Ejemplo:

a=3, b=2;
Sí a=3 **y** b=2 entonces c=a+b; #Hemos usado un operador 'and'
¿Cuánto vale c?

Y no nos olvidemos de los relacionales:

```
< → menor que
> → mayor que
== → igual
<= →  menor o igual que
>= →  mayor o igual que
!= →  distinto de
```

No confundamos el '=' de asignación con el '==' de comparación.

A=B significa que a la variable A se le asigna el valor de B.

ENTRADA / SALIDA

Vamos a ver las dos funciones de entrada y salida que Python nos da, por una parte *print*, su función es mostrar por pantalla; y por otra *raw_input* que recoge caracteres por teclado.

print

Las funciones más importantes y a la vez básicas de cualquier lenguaje son sin duda las de introducción y salida de datos al programa, en Python es bastante sencillo.

Salida por pantalla (print) de una oración o variable se hace de la siguiente manera.

```
#definimos la variable
a = 5
#salida por pantalla por print
print 'esto saldrá por pantalla con el valor ', a
```

Si nos fijamos en este pequeño programa, al principio declaramos la variable a, que en este caso vale 5, y después con la función de salida por pantalla (print) escribimos entre comillas la cadena de caracteres que queremos que se muestre, y después utilizamos una coma para mostrar seguidamente la variable. Por pantalla saldrá:

>> esto saldrá por pantalla con el valor 5

Si quisiéramos que mostrase mas de una variable podríamos hacerlo de dos modos. Ejemplo:

1. Queremos mostrar el mensaje anterior con nombre y apellido:

```
      #definimos las variables nombre y apellido
      nombre='pepe'
      apellido='utiel'
      #Mostramos las variables por pantalla
      print 'Nombre:', nombre,'Apellido:', apellido
```

La salida será:

```
>> Nombre: pepe Apellido: utiel
```

2. Lo mismo pero de otra manera

```
      #definimos las variables
      nombre='pepe'
      apellido='utiel'
      #mostramos por pantalla
      print 'Nombre: %s Apellido: %s' %(nombre,apellido)
```

La salida será:

```
>> Nombre: pepe Apellido: utiel
```

Lo que hacemos con %s es decir que hay ponga una cadena (string) con el valor de la variable que después indicamos, las variables deben declararse al final en el mismo orden que se han puesto en el print.

raw_input()

La otra función de la programación es la de recoger datos mediante el teclado, esta función permite al usuario introducir datos al programa, cuando la lectura del programa llega a un *raw_input()* el programa se detiene esperando una introducción por el usuario. El código sería de la siguiente manera:

```
print '¿Cómo te llamas? '
nombre = raw_input()
print 'Encantado de conocerte ', nombre
```

El programa te pregunta tu nombre y se detiene esperando que teclees una cadena de caracteres, una vez escribas y le des a intro guardará en la variable 'nombre' lo que hayas introducido y utilizará un print como el que hemos explicado. El programa se mostraría en pantalla de la siguiente forma:

```
>> ¿Cómo te llamas?
Manolo
>> Encantado de conocerte Manolo
```

Estructuras de control

Cuando estamos diseñando el algoritmo de lo que va a hacer nuestro programa hay que tener claro lo que queremos y vamos a hacer, y tenerlo todo plasmado en papel antes de pasar a programarlo, la parte más importante son las estructuras.

Estructura Secuencial

```
Instrucción_1
Instrucción_2
Instrucción_3
...
Instrucción n
```

Esta es la estructura más simple, por cada línea hace una instrucción diferente, en acabar todas las instrucciones siguiendo el orden riguroso desde arriba hasta abajo habrá acabado el programa.

Estructura Alternativa Simple

```
if condición :
      Acción1
else:
      Acción2
```

Esta estructura también es muy básica, pero sin duda una de las que más se utiliza, es la condicional. Si ocurre "esto" entonces ir a "esta" instrucción, sino entonces ir a "esta otra".

Estructura Alternativa Múltiple

```
if condición :
      Acción1
elif condición:
      Acción2
elif condición:
      Acción3
...
else:
      AcciónN
```

Es muy similar a la anterior, pero da más opciones a la hora de elegir la condición utilizando "elif".

Estructura Repetitiva

```
while condición:
      sentencias
```

Es el "mientras", se utiliza como: "mientras la condición sea X" entonces hacer "esto". En acabar de hacer "esto", comprobará si la condición se sigue cumpliendo, en caso afirmativo volverá a hacer "esto".

Esta construcción puede provocar bucles infinitos si la condición se cumple siempre.

Repetición hasta Contador

```
for contador in range(fin):
      # sentencias
```

Este bucle se ejecutara *fin* veces. A cada vuelta el valor de la variable contador se incrementa en una unidad. El valor mínimo de la variable contador (Primera vuelta) será 0, el máximo *fin-1* (Ultima vuelta).

```
for contador in range(inicio,fin):
      # sentencias
```

Este bucle se ejecutara *fin-inicio* veces. A cada vuelta el valor de la variable contador se incrementa en una unidad, hasta *fin-1*.

```
for contador in range(fin,inicio,-1):
      # sentencias
```

De esta manera hacemos que el contador decrezca, es importante ver que va primero la variable *fin* y después *inicio*. La variable *fin* ha de ser mayor que *inicio*. Si esto no es así, el bucle no se ejecutara ninguna vez. A cada vuelta el valor de la variable contador se decrementa en una unidad, desde *fin-1*, hasta *inicio*.

Funciones

Las funciones son muy útiles para la programación por módulos, y así poder tener definido un algoritmo en el programa al cual podamos llamar cuando queramos y que haga una serie de instrucciones sin tener que volver a escribir todo el algoritmo.

Antes del programa principal hay que escribir una función en caso de que la vayamos a utilizar, y la estructura de programación es la siguiente.

```
def nombredelafuncion(valores):
      # acciones con los valores
      return valores
```

Y la llamada desde el programa sería...

```
Valor = 1 # un numero entero
nombredelafuncion( valor )
```

El valor a utilizar en la llamada a la función tenemos que declararlo antes.

NOTA: El valor de la llamada a la función no tiene porqué ser igual al definido en la función.

Ejemplo:

```
#Definiendo la función (modulo)
def numMayor (n1,n2):
      if  n1>n2:
            max=n1
      else:
            max=n2
      return max
#programa principal
print 'Introduce un numero: '
num1 = int(raw_input())
print 'Introduce otro numero: '
num2 = int(raw_input())
mayor=numMayor(num1,num2)
print 'El mayor es', mayor
```

En este programa el valor del modulo lo devolvemos con "return", que resulta muy útil para los condicionales o para cuando no es una expresión el valor a devolver sino una condicional o algo por el estilo.

Tablas

La utilidad de utilizar tablas en PYTHON es muy importante para ordenar los datos de salida y entrada de forma fácil y sencilla.

Vamos a explicar algo muy superficial sobre las tablas, ya que esto tiene muchísima historia y hay manuales donde lo explican mucho más extensamente.

Para crear una tabla o matriz nula, utilizamos el siguiente código:

```
Matriz = []
```

Si por el contrario queremos poner algunos valores en la tabla, los ponemos separados por comas, y eso serán las posiciones.

```
Matriz = [3, 4, 5]
```

Si ponemos esto se habrá creado una tabla con 3 posiciones, que PYTHON las tomará como posición 0, 1 y 2. En vez de números también podemos poner nombres o cadenas de caracteres entre "comillas".

```
Matriz = ["pepe", "juan", "carlos"]
```

Una vez creada la matriz podemos incluir más celdas con el siguiente código:

```
Matriz.append("jorge")
```

Esto habrá creado una cuarta celda (posición 3) en la tabla "Matriz". Para visualizar el contenido de la tabla utilizamos...

```
print Matriz
print Matriz[0]
print Matriz[3]
```

>> ['pepe', 'juan', 'carlos', 'jorge']
>> pepe
>> jorge

Si lo que queremos es que el usuario introduzca por teclado los valores de la tabla, utilizaremos el siguiente programa.

Ejemplo:

```
pila = []
pila.append(raw_input())
pila.append(raw_input())
pila.append(raw_input())
pila.append(raw_input())
print pila
pila.pop()
print pila
```

Este programa crea una matriz, y va pidiendo al usuario valores cualquiera ya que no está definido el tipo de valor, una vez le ha pedido los 4 valores, los muestra por pantalla en una matriz.
Posteriormente descarga el ultimo valor de la pila pop() y la muestra de nuevo

Registros

PYTHON no da soporte nativo a los registros, lo simula a través de un modulo llamado record.

```
from record import record
class Persona(record):
  nombre=''
  apellido=''
  edad=0

persona1=Persona(nombre='Juan', apellido='Roca',edad=24)
persona2=Persona(nombre='Ana', apellido='Martinez',edad=19)

persona1.nombre
```

Así se haría un registro en PYTHON, y para mostrar los valores de los registros ponemos el nombre de la tabla, y el campo, en nuestro caso "persona1" es el nombre de la tabla y "nombre" es el campo, por lo que este programa mostrará:

>> Juan

NOTA: Es posible que en tu intérprete PYTHON no te funcionen los registros si no acepta el modulo de record, ya que depende del intérprete.

Ficheros

PYTHON también tiene soporte para ficheros, las 2 funciones más conocidas son Lectura y Escritura de ficheros:

- **Lectura**

  #Paso 1:

  ```
  #abrir el fichero
  fichero=open('texto.txt','r')
  ```

  Mete la información del archivo texto.txt en la variable fichero, el parámetro "r" indica read (leer).

  #Paso 2:

  ```
  #Lee la información
  for linea in fichero:
        print linea
  ```

  Muestra por pantalla la información del archivo.

  #Paso 3:

```
#Cerrar el fichero
fichero.close()
```

Cierra el fichero, es una instrucción obligatoria.

- **Escritura**

#Paso 1:

```
#Abrir fichero
fichero=open('notas.txt','w')
```

Abre el fichero notas.txt, y lo asigna a la variable fichero, con el parámetro "w" (write) que permite la escritura.

#Paso 2:

```
#Escribir la información
for in range(10):
      nota=rae_input('Nota: ')
      fichero.write(nota+ '\n')
```

Esto te pregunta hasta 10 líneas 10 notas, una vez acaba el usuario de escribir las 10 líneas sale del bucle.

#Paso 3:

```
#Cerrar el fichero
fichero.close()
```

Cierra el fichero.

Documentación oficial Python

Guía de aprendizaje de Python (Español):
http://es.tldp.org/Tutoriales/Python/Tutorial-Python/

Manual Python (Ingles):
http://docs.python.org/

Agradecimientos

Quiero agradecer la ayuda prestada cuando me estaba iniciando en PYTHON del canal del IRC-Hispano #PYTHON, especialmente a **brainsuker** y **pizte**.

Esta última versión del manual ha sido modificada por **Philex** y por **brainsucker**.

Muchas Gracias por vuestra atención.