Introducción a la Programación en Python

José Ortiz Bejar

Cadenas, Listas y Archivos

job@correo.fie.umich.mx

Universidad Michoacana de San Nicolás de Hidalgo

28 de agosto de 2014

Introducción

- Está sección describe
  - Cadenas (Strings)
  - Listas
  - Archivos (File I/O)

Strings

- Una cadena es una secuencia inmutable de caracteres
- Secuencia significa que puede ser indexada
  - Los índices comienzan en 0 (como en C, Java y C#)
  - Entonces text[0] es el primer caracter de text
- La función (built-in) **len** regresa la longitud de una cadena
  - El último caracter de text está en el índice *len(text)-1*

Strings

```
element = "boron"
i = 0
while i < len(element):
    print element[i]
    i += 1
#------Salida------
b
o
r
o
n
```

- **Nota**: No existe un tipo separado para el tipo caracter. Un caracter es una cadena de longitud 1

Inmutabilidad

- Inmutable significa que no puede ser modificada una vez que ha sido creada (e.i.) No se pueden modificar los caracteres individuales

```
$ python
>>> element = 'oro'
>>> print 'elemento es', element

elemento es oro

>>> element[0] = 's'

TypeError: object does not support item assignment
```

Inmutabilidad

- Porqué?
  - Seguridad (La discutiremos más adelante)
  - Es muy costoso cambiar la longitud de una cadena.
- Si es posible asignar una nueva cadena a una variable

```
element = 'oro'
print 'elemento is', element
element = 'plomo'
print 'elemento ahora es ', element

elemento es oro
elemento ahora es plomo
```

División (Slicing)

- text[inicio:fin] toma una sub-cadena del texto
  - Crea una nueva cadena que contiene los caracteres desde inicio hasta fin (no incluye el caracter en el índice fin)

```
element = "hidrogeno"
print element[1:3], element[:2], element[4:]

id hi ogeno
```

Verificación de límites

- Python siempre verifica los limites cuando se accede por índice a un solo elemento.
- Pero trunca cuando se toma un intervalo

```
$ python
>>> element = 'helium'
>>> print element[1:22]

elium

>>> x = element[22]

IndexError: string index out of range
```

Índices negativos

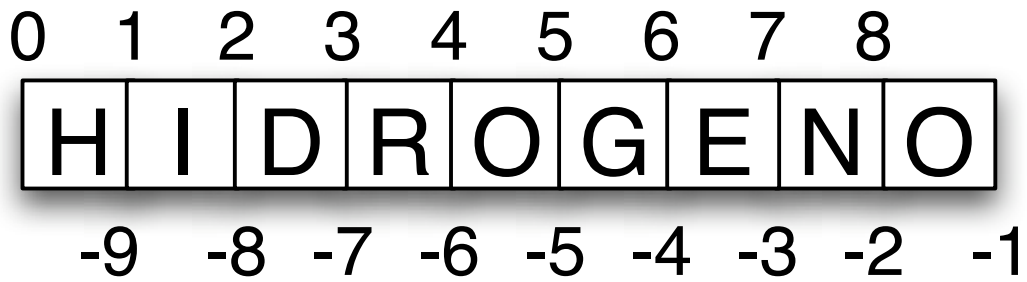
- Con índices negativos es posible recorrer la cadena en sentido inverso
  - x[-1] es el último caracter
  - x[-2] es el penúltimo caracter

```
element = "carbon"
print element[-2], element[-4], element[-6]
```

```
o r c
```

Índices negativos

- Nos puede ayudar ver los índices como si estuvieran entre los caracteres



Consecuencias

- text[1:2] es cualquiera de:
  - El segundo caracter en text
  - La cadena vacía si text no tiene segundo caracter
- text[2:1] es siempre la cadena vacía
- text[1:1] desde la posición 1 pero sin incluir 1, es la cadena vacia
- text[1:-1] Es toda la cadena menos el primer y ultimo caracter, lo cual puede ser la cadena vacía

Métodos

- Un método es una función que es parte de un objeto particular
  - Ayudan a organizar el código
  - En secciones posteriores revisaremos como crear objetos
- Casi todo en python tiene métodos
  - Los números son la única excepción
- Para llamar un método *meth* del objeto *obj* se ejecuta escribiendo obj.meth()

Métodos de los objetos String

Método	Propósito	Ejemplo	Resultado
capitalize	Capitaliza la primera letra	"text".capitalize()	"Text"
lower	Convierte todas las letras a minúsculas.	"aBcD".lower()	"abcd"
upper	Convierte todas las letras a mayúsculas.	"aBcD".upper()	"ABCD"
strip	Remueve los en blanco espacios al inicio y final de la cadena (tabulador, saltos de linea, etc.)	" a b ".strip()	"a b"
lstrip	Remueve los espacios en blanco que se encuentran del lado izquierdo de la cadena	" a b ".lstrip()	"a b "
rstrip	Remueve los espacios en blanco que se encuentran del lado derecho de la cadena	" a b ".rstrip()	" a b"
count	Cuenta cuantas veces aparece una cadena en otra	"abracadabra".count("ra")	2
find	Regresa el índice de la primera ocurrencia de una cadena en otra o -1.	"abracadabra".find("ra") "abracadabra".find("xyz")	2 -1
replace	Reemplaza las ocurrencia de una cadena por otra	"abracadabra".replace("ra", "-")	"ab-cadab-"

Encadenamiento de llamadas a métodos

- Las llamadas a métodos pueden encadenarse
  - Si el resultado de una llamada es un objeto, puede inmediatamente llamarse un método del nuevo objeto.

```
element = "aluminio"
print ':' + element.upper()[4:7].center(10) + ':'

:    INI    :
```

- Debe usarse esto con moderación. Largos encadenamientos pueden resultar difíciles de leer y depurar.

Pruebas de pertenencia

- Se usan para probar si una cadena aparece en otra
  - Es más simple que el método find
  - No indica en que posición inicia la ocurrencia

```
print "geno" in "hidrogeno"
print "genio" in "hidrogeno"

True
False
```

Listas

- Una lista es una secuencia mutable de objetos.
  - Mutable significa que, a diferencia de las cadenas puede ser modificada sin tener que crear una copia.
  - Puede contener cualquier tipo de datos
  - Es semejante a un array o un vector, que se redimensiona automáticamente
- Se crean listas poniendo los valores entre paréntesis cuadrados
  - La lista vacía se escribe []

Listas

- Se utilizan índices y se pueden dividir como las listas
  - Al igual que en las listas, Python verifica los límites cuando se usan índices, pero trunca cuando se segmenta

```
gases = ['He', 'Ne', 'Ar', 'Kr']
print gases
print gases[0], gases[-1]

['He', 'Ne', 'Ar', 'Kr']
He Kr
```

Modificación de listas

- Usando x[i] = v se asigna un nuevo valor a un elemento de la lista.

```
gases = ['He', 'Ne', 'Ar', 'Kr']
print 'before:', gases
gases[0] = 'H'
gases[-1] = 'Xe'
print 'after:', gases
#--------------Salida--------------
before: ['He', 'Ne', 'Ar', 'Kr']
after: ['H', 'Ne', 'Ar', 'Xe']
```

Modificación de listas

- El elemento ya debe existir

```
$ python
>>> gases = ['He', 'Ne', 'Ar', 'Kr']
>>> print 'before:', gases

before: ['He', 'Ne', 'Ar', 'Kr']

>>> gases[10] = 'Ra'

IndexError: list assignment index out of range
```

Modificación de listas

- Se usa **append** para agregar elementos al final de la lista.
  - La lista crece conforme se necesita

```
vocales = []
print vocales
for c in 'aeiou':
    vocales.append(c)
    print vocales
#------Salida-------
[]
['a']
['a', 'e']
['a', 'e', 'i']
['a', 'e', 'i', 'o']
['a', 'e', 'i', 'o', 'u']
```

Concatenación

- Las suma de cadenas (o listas) crea una nueva cadena (o lista) con todo el contenido de los originales

```
element = 'carbon'
masa = '14'
print elemento + '-' + masa

lantanidos  = ['Ce', 'Pr', 'Nd']
actinidos = ['Th', 'Pa', 'U']
todos = lantanidos + actinidos
print todos

carbon-14
['Ce', 'Pr', 'Nd', 'Th', 'Pa', 'U']
```

Concateneción

- No es posible concatenar una cadena con una lista
  - Pero **list(text)** crea una lista donde los elementos son los caracteres de text.

```
water = 'H2O'
print 'before conversion:', water
water = list(water)
print 'after conversion:', water
#-------salida-------
before conversion: H2O
after conversion: ['H', '2', 'O']
```

Borrado de elementos

- **del** borra un elemento de una lista
  - Acortar unas lista puede ocasionar problemas si se está iterando sobre ellas en ese momento.

```
organicos = ['H', 'C', 'O', 'N']
print 'original:', organicos
del organicos[2]
print 'despues de borrar el elemento 2:', organicos
del organicos[-2:]
print 'despues de borrar los dos ultimos:', organicos
#-----salida-------
original: ['H', 'C', 'O', 'N']
despues de borrar el elemento 2: ['H', 'C', 'N']
despues de borrar los dos ultimos elementos: ['H']
```

Modificación de listas

- Es posible borrar segmentos

```
organicos = ['H', 'C', 'O', 'N']
print 'original:', organicos
del organicos[1:-1]
print 'despues de borrar:', organicos

#------- salida --------
original: ['H', 'C', 'O', 'N']
despues de borrar: ['H', 'N']
```

- **Nota:** del es una sentencia, no un operador. No regresa la lista modificada.

Métodos de listas

- En el ejemplo, metales inicialmente tiene ['oro', 'hierro', 'plomo', 'oro']

Método	Propósito	Ejemplo	Result
append	Agrega un elemento al final	metales.append('zinc')	['oro', 'hierro', 'plomo', 'oro', 'zinc']
index	Regresa el índice del elemento	metales.sort('plomo')	2
count	Cuenta cuantas veces aparece un elemento.	metales.count('oro')	2
find	Encuentra la primera ocurrencia de un elemento	metales.find('hierro') metales.find('mercurio')	1 -1
insert	Inserta un elemento en una posición dada	metales.insert(2, 'plata')	['oro', 'hierro', 'plata', 'plomo', 'oro']
remove	Borra la primera ocurrencia de un elemento.	metales.remove('oro')	['hierro', 'plomo', 'oro']
reverse	Invierte la lista en sitio.	metales.reverse()	['oro', 'plomo', 'hierro', 'oro']
sort	Ordena la lista en sitio	metales.sort()	['hierro', 'oro', 'oro', 'plomo']

Notas acerca de los métodos en listas

- *index* reporta un error si el elemento no se encuentra
- *reverse* y *sort* modifican la lista y regresan None
  - Es equivalente a cero
  - Al igual que 0 y la cadena vacías, es equivalente a False
- x = x.reverse() es un erro común, en el que x es puesta a None, por tanto los datos se pierden.

Ciclos for

- Los ciclos **for** en Python iteran sobre el contenido de una colección (listas, cadenas ...)
  - *for c in alguna-cadena* asigna c cada caracter de alguna-cadena
  - *for v in alguna-lista* asigna v a cada valor de alguna-lista
  - Se puede utilizar cualquier nombre para la variable (c,v)

Ciclos for

```
for c in 'plomo':
    print '/' + c + '/',
print

for v in ['he', 'ar', 'ne', 'kr']:
    print v.capitalize()
#-----salida-------
/p/ /l/ /o/ /m/ /o/
He
Ar
Ne
Kr
```

- Es usualmente lo que se hace con un ciclo for

Rangos

- La función range crea la lista [inicio, inicio+1 ..., fin-1]
  - fin-1 es consistente con x[inicio:fin]
  - Los caso especial range(fin) y range(inicio,fin,inc)
  - range también puede generar una lista vacía

Rangos

- La función range crea la lista [inicio, inicio+1 ..., fin-1]

```
print 'up to 5:', range(5)
print '2 to 5:', range(2, 5)
print '2 to 10 by 2:', range(2, 10, 2)
print '10 to 2:', range(10, 2)
print '10 to 2 by -2:', range(10, 2, -2)
#-------salida-----------------
up to 5: [0, 1, 2, 3, 4]
2 to 5: [2, 3, 4]
2 to 10 by 2: [2, 4, 6, 8]
10 to 2: []
10 to 2 by -2: [10, 8, 6, 4]
```

Iteración sobre rangos

- Un ciclo de 0 a N-1, usa *for i in range(N)*
- Un ciclo sobre los índices de una lista o cadena, usa *for i in range(len(sequence))*

```
element = 'plomo'
for i in range(len(element)):
    print i, element[i]
#----salida-----
0 p
1 l
2 o
3 m
4 o
```

Pertenencia

- x in c opera la lista elemento a elemeto
  - Entonces 3 in [1,2,3,4] es True
  - Pero [1, 2, 3, 4] es False

Listas por compresión

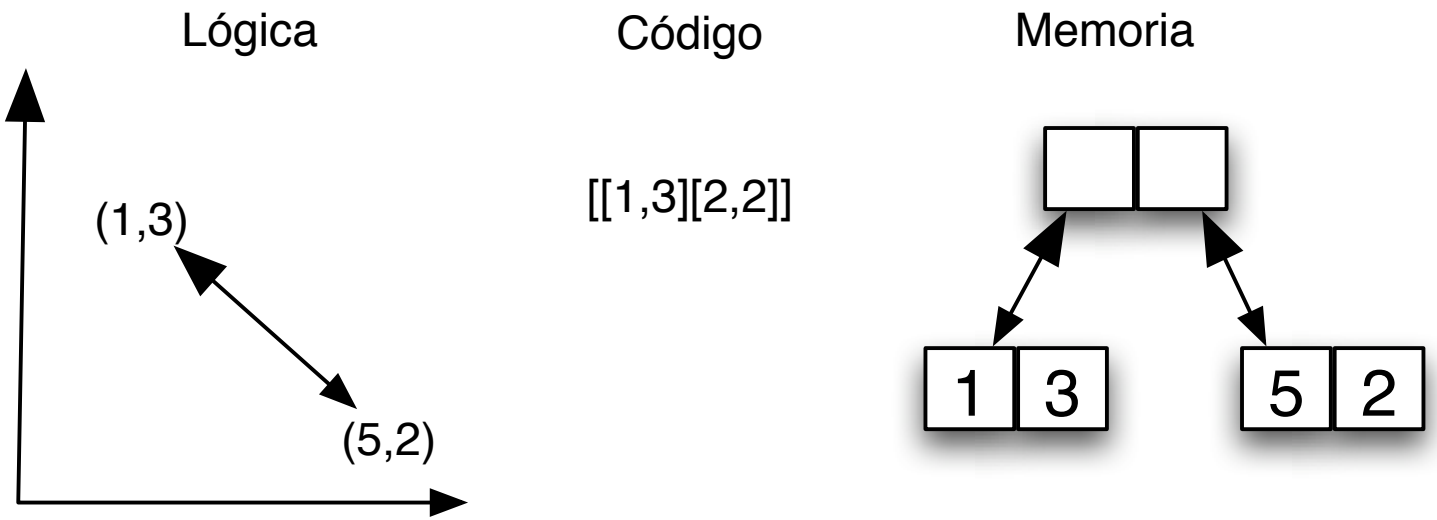
Permite crear un lista de forma natural(matematicamente).

- $C = \{x | x^2 \text{ en } \{0 \ldots 9\}\}$
- $P = \{x | x \text{ en } \{1, 2, 4 \ldots 2^{12}\}\}$
- $E = \{x | x \text{ en } C \text{ and } x \text{ es par } \}$

```
>>> C = [x**2 for x in range(10)]
>>> P = [2**i for i in range(13)]
>>> E = [x for x in C if x % 2 == 0]
```

Listas anidadas

- Las listas pueden contener otras listas



Listas anidadas

- Los índices de izquierda a derecha seleccionan los elementos de afuera hacia adentro

```
elements = [['H', 'Li', 'Na'], ['F', 'Cl']]
print 'Primer elemento de la lista:', elements[0]
print 'Segundo elemeto de la segunda sublista:',
print elements[1][1]

 #--------salida-----------
Primer elemento de la lista: ['H', 'Li', 'Na']
Segundo elemeto de la segunda sublista: Cl
```

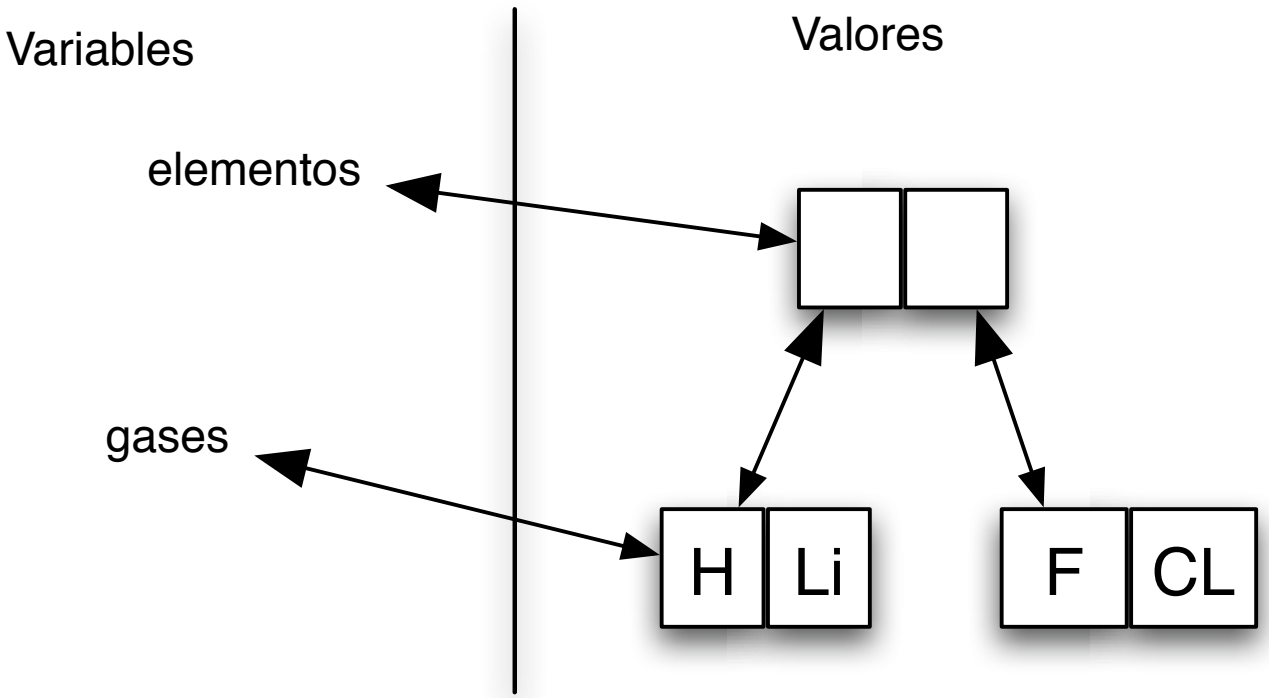
Alias

- Las listas anidadas son objetos por si mismos
  - Las listas exteriores almacenas referencias a las listas interiores
  - Pero las listas interiores no saben que también pueden ser referenciadas
- Subscripting las listas exteriores se crean alias a la interiores
  - Otro nombre para los mismos datos
- Los cambios hechos a la referencia actualizan los datos origiales

Alias

```
elementos = [['H', 'Li'], ['F', 'Cl']]
gases = elementos[1]
print 'antes \n'+'elementos:', elementos
print 'gases:', gases
gases[1] = 'Br'
print 'despues \n'+'elementos:', elementos
#-----salida-----------------
antes
elementos: [['H', 'Li'], ['F', 'Cl']]
gases: ['F', 'Cl']
despues
elementos: [['H', 'Li'], ['F', 'Br']]
```

Alias



XXXXXXX

Índices vs Slicing

- Índices y slicing regresan diferentes tipos de cosas para listas
  - Mediante el índice se regresa una referencia al elemento
  - Slicing regresa una nueva lista que contiene los elementos originales.
- Cambiar un slice no afecta la lista original

Índices vs Slicing

```
metales = ['Cr', 'Mn', 'Fe', 'Co', 'Ni', 'Cu', 'Zn']
mitad = metales[2:-2]
print 'antes\n'+'metales:', metales, '\n mitad:', mitad
mitad[0] = 'Al'
del mitad[1]
print 'despues\n'+'metales:', metales, '\n mitad:', mitad
#-------salida-------
antes
metales: ['Cr', 'Mn', 'Fe', 'Co', 'Ni', 'Cu', 'Zn']
mitad: ['Fe', 'Co', 'Ni']
despues
metales: ['Cr', 'Mn', 'Fe', 'Co', 'Ni', 'Cu', 'Zn']
mitad: ['Al', 'Ni']
```

Índices vs Slicing

- Note que la copia solo llega a un nivel de profundidas
- Como las cadenas son inmutables siempre se copian
- Es útil hacer dibujos cuando hay varios niveles.
  - Si las figura son complicadas, se debe simplificar el código

Tuples

- Python tiene un segundo tipo de lista, llamado tuples
  - A diferencia de las listas, estos son inmutables
  - Se escriben utilizando paréntesis en lugar de paréntesis cuadrados: (1, 2, 3) en lugar de [1, 2, 3]
  - () es el tuple vacío
  - Un tuple con un elemento debe escribirse con coma, como en (55,)
    - Debido a que (55) es el entero 55
- Por qué? Debido a que en ocaciones es necesario conocer una secuencia de valores

Asignación múltiple de valores

- No son necesarios los paréntesis para los tuples
  - 1,2,3 es lo mismo que (1,2,3)
- Permite asignación múltiple
  - izq, der = 'oro', 'plomo' asigna 'oro' a izq y 'plomo' a der
  - izq, der = ('oro', 'plomo')
- Python convierte lista a tuples si es necesario
  - izq,mitad,der = ['oro',hierro, 'plomo'] funciona
  - izq,mitad,der = ['oro','hierro', 'plomo'] funciona
    - el número de variables debe coincidir con el número de valores
- Intercambio de valores
  - izq, der = der, izq realiza un intercambio seguro

Desempaquetando estructuras en ciclos

- Se puede utilizar la asignación múltiple en ciclos para desempaquetar estructuras al vuelo

```
elementos = [
    ['H', 'hidrogeno', 1.008],
    ['He', 'helio', 4.003],
    ['Li', 'litio', 6.941]]
for (simbolo, nombre, peso) in elementos:
    print nombre + ' (' + simbolo + '): ' + str(peso)
#-------salida----------
hidrogeno (H): 1.008
helio (He): 4.003
litio (Li): 6.941
```

Desempaquetando esctructuras

- Dos razones por la que los lenguajes interpretados son productivos son:
  - Permiten escribir estructuras de datos complejas directamente
  - Permiten separarlas fácilmente.

Archivos

- Utiliza la función **open** para abrir un archivo
  - El primer argumento es la ruta
  - El segundo 'r' (para lectura) o 'w' (para escritura).

```
input_file = open('count_bytes.py', 'r')
content = input_file.read()
input_file.close()
print len(content), 'bytes en el archivo'
#--------salida------------
121 bytes en el archivo
```

Métodos de Archivos

Método	Propósito	Ejemplo
close	Cierra el archivo	input_file.close()
read	Lee N bytes del archivo, regresa una cadena vacía si el archivo esta vacío. Si N no es dado, lee el resto del archivo.	next_block = input_file.read(1024) rest = input_file.read()
readline	Lee una linea del archivo, regresa la cadena vacía si el archivo esta vacío	line = input_file.readline()
readlines	Regresa las lineas restantes de un archivo como liata, o una lista vacía si el archivo esta vacío	rest = input_file.readlines()
write	Escribe una cadena en un archivo. write no agrega un salto de linea	output_file.write(".Elemento 8: Oxigeno")
writelines	Escribe las cadenas de una lista en el archivo (sin agregar saltos de linea).	output_file.writelines(["H", "He", "Li"])

Copiado de archivos

```
input_file = open('file.txt', 'r')
output_file = open('copy.txt', 'w')
line = input_file.readline()
while line:
    output_file.write(line)
    line = input_file.readline()
input_file.close()
output_file.close()
```

Ciclos sobre archivos

- Los ciclos en archivos regresan lineas de texto(pueden ser bytes o caracteres) incluyendo el salto de linea y el retorno de carro.

```
input_file = open('count_lines.py', 'r')
count = 0
for line in input_file:
    count += 1
input_file.close()
print count, 'lines in file'

6 lines in file
```

Otras formas de copiar archivos

```
input_file = open('file.txt', 'r')
lines = input_file.readlines()
input_file.close()

output_file = open('copy.txt', 'w')
output_file.writelines(lines)
output_file.close()
```

- Leer todas las lineas en una lista
- Escribir las lineas en la lista en el archivo de salida

Otra más

```
input_file = open('file.txt', 'r')
output_file = open('copy.txt', 'w')
for line in input_file:
    line = line.rstrip()
    print >> output_file, line
input_file.close()
output_file.close()
```

- print ¿¿ output_file envía la salida de print al archivo
- Automáticamente se agrega la linea la final del archivo

Sumario

- Las características básicas de los lenguajes de programación modernos son las mismas
  - Strings, Listas, file I/O
- La diferencia es como las presentas
  - Las sintáxis de Python es clara y consitente