

Comprensión de listas (I)

Python permite crear a una lista con todos los valores en una única instrucción.

Es más rápido que hacerlo valor a valor.

<lista> = [<expr> **for** <var> **in** ...]

li = [('a', 1), ('b', 2), ('c', 7)]

[n * 3 **for** (x, n) **in** li]

Comprensión de listas (II)

- Con bucles:

y = []

for i in range(n+1) :

 y.append(a+i*h)

- Con comprensión de listas:

v = [a+i*h **for** i **in** range(n+1)]

- Otro ejemplo:

gradosC = [convertirF_C(i) **for** i **in** gradosF]

Comprensión de listas (III)

- Listas más complejas

```
L = [[0 for col in range(5)] for fil in range(6)]
```

```
tabla2 = []
```

```
for C, F in zip(gradosC, gradosF) :
```

```
    fil = [C, F]
```

```
    tabla2.append(fil)
```

- Alternativa

```
tabla2 = [[C, F] for C, F in zip(gradosC, gradosF)]
```

Comprensión de listas (IV)

```
nums = [1,2,3,4]
```

```
frutas = ["Manzanas", "Melocotones", "Peras",  
"Plátanos"]
```

```
print ([ (i,f) for i in nums for f in frutas])
```

```
list1 = [3,4,5]
```

```
mult = [item*3 for item in list1]
```

```
palabras = ["esto","es","una","lista","de","palabras"]
```

```
items = [ palabra[0] for palabra in palabras ]
```

Operaciones adicionales

- `enumerate(L)` devuelve un par (indice,elem)
`for i, c in enumerate(L) :`
$$L[i] = i + c$$
- `sorted(L)` devuelve una lista ordenada
- `reversed(L)` devuelve una lista con los elementos en orden inverso

Cuestiones sobre listas y referencias

- ¿qué ocurre en este caso?

`L = [[0, 0]] * 5`

`L[2][0] = 7`

- ¿y en este otro?

`L = ["abc"] * 3`

`L = [L] * 5`

`L[2][0] = 'cde'`

Conversión de str a list

Hay que tener cuidado porque

`l = []`

`s = "Hola"`

`l += s`

o `l = list("Hola")`

No da como resultado una lista con un único elemento “Hola” sino

`['H', 'o', 'l', 'a']`

Resumen de Operaciones sobre listas (I)

- `a = []` crea una lista vacía
- `A = [1.3, 4, 'script.py']` asigna un valor
- `a.append(elem)` añade un elemento al final
- `a + [1,3]` concatena dos listas
- `a[3]` accede a un elemento
- `a[-1]` accede al último elemento
- `a[1:3]` devuelve una sublista
- `del a[3]` elimina un elemento

Resumen de Operaciones sobre listas (II)

- `a.remove(4.4)` elimina un elemento con ese valor.
- `a.index('run.py')` devuelve la posición donde se encuentra un elemento (**ValueError** si no está).
- `a.find('run.py')` devuelve la posición donde se encuentra un elemento (-1 si no está).
- `a.count(v)` cuenta el número de veces que aparece un elemento (en este caso no hay error).
- `len(a)` devuelve el número de elementos.
- `min(a)` devuelve el elemento más pequeño.
- `max(a)` devuelve el elemento mayor.
- `sum(a)` suma todos los elementos.

© Prof. Miguel García Silvente

115

Resumen de operaciones sobre listas (III)

- `a.sort()` ordena una lista.
- `as = sorted(a)` devuelve una versión ordenada.
- `a.reverse()` invierte el orden de los valores de una lista
- `b[3][0][2]` indexado anidado
- `isinstance(a, list)` devuelve True si a es una lista
- `l1.copy()` devuelve una copia de la lista

Ejemplo string + list

- **readline()** Lee una línea completa.
`linea = sys.stdin.readline()`
- **readlines()** Lee una serie de líneas, cada línea va en una posición de la lista.
`texto = sys.stdin.readlines()`
- **split()** divide una cadena y devuelve una lista de subcadenas.
`palabras = cad.split()`
`datos = cad.split("es")`
- **strip()** elimina separadores de ambos extremos.

© Prof. Miguel García Silvente

117

Ejemplo string + list (II)

```
[el,gato,sentado,sobre,el,coche].count(el)
el gato sentado sobre el coche.split().count(el)
palabras = 'el gato sentado sobre el coche'.split()
' el gato '.strip()
' el gato '.lstrip()
```

Ejemplo: leer los logins de un fichero de passwords en un sistema linux.

Ejemplos

primos = [2, 3, 5, 7, 11, 13, 17, 19]

[8 / p for p in primos]

[p for p in primos if p % 3 > 1]

[[0,0,0] for x in range(2,5)]

[93 % p for p in primos if 93 % p != 0]

[5 ** p for p in primos if p % 4 > 2 0]

[[fil[i] for fil in matriz] for i in range(len(matriz))]

L = [[0 for i in range(5)] for j in range(6)]

© Prof.Miguel García Silvente

119

Tipos de datos asociativos

- Conjunto: **set**
- Diccionario: **dict**

Tipo de dato set

- Es una colección no ordenada de datos (pueden ser de distinto tipo).

`s1 = {1,3,5,7,5,4,3,2,1}`

`s2 = {1, 'a' }`

- Debe ser de tipo “hashable” (que se pueda “aplicar una función hash sobre él”)

`>>> s1 = {(1, 'a'), (2, 'b')}`

`>>> conj = {(3,5), (6,2), "Hola"}`

`>>> print (conj)`

`>>> s2 = {[1,2,3], [1,3,4]}`

`set([(6, 2), 'Hola', (3, 5)])`

...

`TypeError: unhashable type: 'list'`

© Prof. Miguel García Silvente

121

Tipos de datos conjunto

- Existe `set` como versión mutable y `frozenset` e `immutableSet` como inmutables.
- Ejemplos:

`dias_semana = {'lun', 'mar', 'mié', 'jue', 'vie'}`

`dias_semana = set('lun', 'mar', 'miércoles', 'jueves', 'viernes')`

`dias_semana.add('sáb')`

`'mié' in dias_semana` devuelve `True`

`dias_semana.remove('mié')` elimina elemento

© Prof. Miguel García Silvente

122

Operaciones sobre conjuntos (I)

- Añadir un elemento: `add(x)`
- Eliminar un elemento: `remove(x)` `discard(x)`

la diferencia entre ambos:

`s1 ={1,3,5,7,5,4,3,2,1}`

`s1.discard(8) # no ocurre nada`

`s1.remove(8) # genera una excepción KeyError: 8`

- Devuelve y elimina un elemento aleatorio: `pop()`
- Copiar todos los elementos: `copy()`
- Eliminar todos los elementos: `clear()`

Operaciones sobre conjuntos (II)

Incluye `union`, `intersection`, `difference`, `symmetric_difference`

`s1 ={1, 3, 5, 7, 5, 4, 3, 2, 1}`

`s2 ={2, 4, 6, 8, 6, 4, 3, 2, 1}`

`union_s1_s2 = s1.union(s2)`

`intersect_s1_s2 = s1.intersection(s2)`

`diff = s1.difference(s2) #set((5,7))`

`s_diff = s1.symmetric_difference(s2) # set((5,6,7,8))`

Iterar sobre un conjunto

En python 3

```
for letra in set("zbatgdab"):  
    print (letra)
```

d
g
z
b
t
a

¿Qué ocurre en python 2??

Comprensión de conjuntos

De forma similar a las listas:

```
>>> a = {x for x in 'abracadabra' if x not in 'abc'}  
set(['r', 'd'])  
  
>>> vocalesPalabra = {x for x in 'esto es una prueba'  
if x in 'aeiou'}  
set(['a', 'u', 'e', 'o'])  
  
>>> L = [34, 45, 87, 90, 32, 4]  
  
>>> mayores50 = {x for x in L if x>50}  
set([90, 87])
```

Tipo de dato dict (I)

Tipo *diccionario*: Permite guardar pares.

(clave, valor) y poder acceder a ellos de forma eficiente a través de la clave.

Las claves deben ser **inmutables**. Los valores no tienen restricciones.

Tipo de dato dict (II)

Se pueden guardar claves y valores de distintos tipos.

Operaciones: buscar, borrar, modificar.

También se les conoce como **tablas hash** o **arrays asociativos**.

Asignación y acceso (I)

- Se usa {}, separando clave y valor con : y cada par con ,
`datos = {'nombre' : 'Pepe', 'edad': 40}`
`datos = dict(nombre='Pepe', edad=40)`
- Se accede con [] usando la clave
`datos['nombre']`
`datos['edad']`
`datos['Pepe'] ?????`
- Internamente usan hashing

© Prof.Miguel García Silvente

129

Asignación y acceso (II)

- Ejemplo: temperaturas de ciudades
`temps = {'Madrid' : 29, 'Granada': 30, 'Valencia' : 28}`
`temps = dict(Madrid=29, Granada=30, Valencia=28)`
- Modificar datos (si ya existe)
`temps['Granada'] = 29`
- Insertar datos (si no existía)
`temps['Málaga'] = 30`

© Prof.Miguel García Silvente

130

Actualización

- Se actualiza usando la clave con []
datos['nombre'] = 'José'
datos['domicilio'] = 'Casa'
datos['id'] = 4532
- Los diccionarios no tienen ningún orden preestablecido.

```
>>> datos
```

```
{'nombre': 'José', 'id': 4532, 'edad': 40,  
'domicilio': 'Casa'}
```

Asignación directa

Ejemplo: temperaturas de ciudades

```
ciudades = ['Madrid', 'Granada', 'Valencia']  
temps_ciudad = [29, 30, 28]
```

```
temps = dict(zip(ciudades, temps_ciudad))
```

```
print (temps)
```

```
{'Madrid': 29, 'Granada': 30, 'Valencia': 28}
```

Borrar datos

- Para borrar un elemento se usa **del**
del temps['Valencia']
- También se puede usar **pop**
temps.pop('Málaga')
- Se pueden eliminar sólo los valores
temps['Granada'] = None
- Para borrar todos se usa **clear**
temps.clear()

© Prof.Miguel García Silvente

133

Iterar y buscar

- Se puede iterar sobre las claves:
for i in temps :
print (i, temps[i])
- Se puede usar **in** para comprobar si una clave está en el diccionario:
if 'Granada' in temps :
print ('Temperatura Granada:',
temps['Granada'])
else :
print ('No hay temperatura para Granada')

© Prof.Miguel García Silvente

134

Obtener claves y valores (I)

Los datos y los valores se pueden consultar como listas:

```
>>> temps.keys()  
['Valencia', 'Granada', 'Madrid']  
>>> temps.values()  
[28, 30, 29]
```

Los elementos no están ordenados, solución: **sorted**

```
for i in sorted(temps.keys()) :  
    temp = d[i]  
    print (temp)
```

Obtener claves y valores (II)

Los datos y los valores se pueden consultar como una lista de pares:

```
for ciudad, temp in temps.items() :  
    print (ciudad, ':', temp)
```

O de forma ordenada:

```
for ciudad, temp in sorted(temps.items()) :  
    print (ciudad, ':', temp)
```

ciudades = temps.keys()

Ejemplo: imprimir contenido

```
estaciones = {'primavera': {'marzo', 'abril','mayo'},  
'verano' : {'junio', 'julio', 'agosto'} }  
for i, j in estaciones.items() :  
    for k in j :  
        print (k)
```

Número de parámetros variable en una función

Se indica con * delante del parámetro

```
def func(*datos) :  
    for i in datos :  
        print (i)  
func(5)  
func(5,6,7)  
func([1,'a', 'c'])
```

Parámetros con nombre en una función

Se indica con ** delante del parámetro

```
def func(**datos) :  
    for i, j in datos.items() :  
        print (i, j)  
func(pepe=1234,juan=34545)
```

Tipo de dato array

- Todos los datos del mismo tipo: módulo array
- Se crean indicando el tipo de dato

```
array(<tipo>, <datos>)
```

```
>>> arr = array(`l`, [1, 2, 3, 4, 5])
```

```
>>> arr [0:3]
```

```
[1, 2, 3]
```

```
>>> arr[3:-1]
```

```
[4, 5]
```

```
>>> arr[-1:0:-1]
```

```
[5, 4, 3, 2]
```

Códigos de tipos de datos

Código	Tipo C	Tipo Python	Tamaño mínimo en bytes
'c'	char	character	1
'b'	signed char	int	1
'B'	unsigned char	int	1
'u'	Py_UNICODE	Unicode char	2
'h'	signed short	int	2
'H'	unsigned short	int	2
'i'	signed int	int	2
'T'	unsigned int	long	2
't'	signed long	int	4
'L'	unsigned long	long	4
'f'	float	float	4
'd'	double	float	8

Tipo de dato fichero

- Un fichero es una secuencia de bytes almacenada en memoria externa.
- Python maneja los archivos como secuencias de caracteres. Sólo se leen y escriben cadenas.
- Es necesario abrir el fichero para poder usarlo y posteriormente hay que cerrarlo.

Operaciones sobre ficheros (I)

- Abrir un fichero (debe tenerse permiso):
`open(<nombre>, <modo>)`
donde `<modo>` puede ser r, w, a
`f = open("datos.txt")`
se deben tener permisos suficientes.
- Cerrar un fichero.
`f.close()`
- Comprobar si existe un fichero.
`os.path.isfile(<nombre>)`

© Prof. Miguel García Silvente

143

Operaciones sobre ficheros (II)

- Comprobar permisos sobre el fichero:
`os.stat(<nombre>)`
- Leer un número de bytes de un archivo:
`f.read(<num>)`
si no se indica un número, se lee hasta el final.
- Leer una línea completa:
`f.readline()`
- Leer todo el archivo por líneas (en una lista):
`f.readlines()`

© Prof. Miguel García Silvente

144

Operaciones sobre ficheros (III)

- Para posicionarse sobre un byte concreto:

`seek(<num>, <donde>)`

donde puede ser `0:inicio, 1:actual, 2:final`

- Conocer la posición actual en el fichero:

`tell()`

- Iterar sobre las líneas del fichero (como una lista)

`for linea in fich :`

- Como un bloque:

`with open("fichero.txt", "rb") as f:`