Tipos de datos estructurados

Secuencias inmutables
Strings, range, tuplas
Secuencias mutables
Listas

UNIVERSITAT POLITÈCNICA DE CATALUNYA BARCELONATECH

Ferran Juan Baruel
Ciències de la Computació / 2017

Tipos de datos

Tipos de datos

Mutabilidad de datos

- Inmutables o estáticos
  - De contenido homogéneo , str() y range()
  - De contenido heterogéneo , tuple()
- Mutables o dinámicos
  - Tanto homogéneos como heterogéneos:
    - Listas de datos (list)
    - Diccionarios (dict)

Secuencias de datos inmutables. Tuplas

- Tuplas o secuencias de datos:
  - Se muestran siempre como una secuencia de datos separados por comas y entre paréntesis(opcional)
  - Las funciones que devuelven más de un valor, devuelven una tupla con el conjunto de valores.
  - Se puede definir una tupla con la función tuple()
  - Se pueden añadir elementos pero no modificarlos

```
>>> a = tuple()
>>> b = ()
>>> type(a)
<class 'tuple'>
>>> type(b)
<class 'tuple'>
>>>
```

```
>>> a = (2,3,4)
>>> a += (5,6)
>>> a
(2, 3, 4, 5, 6)
>>> a += (1,)
>>> a
(2, 3, 4, 5, 6, 1)
```

Funciones comunes

- Funciones comunes a todas las secuencias, tanto mutables como inmutables (#1):

Operación	Resultado
x [not] in s	True si x se encuentra dentro de la secuencia s False en caso contrario.
s + t	Si s y t son secuencias, se concatena una con otra
s * n or n * s	Equivalente a añadir la secuencia s n veces
len(s)	Longitud de s
min(s)	El valor más pequeño de la secuencia s
max(s)	El valor más grande de la secuencia s
sum(s)	Si s es una secuencia numériuca, calcula la suma de sus elementos
s.count(v)	Devuelve el numero de apariciones del valor v en s
s.index(x[, i[, j]])	Índice de la primera ocurrencia de x en la secuencia s (o después de índice i hasta el índice j)

#1 : https://docs.python.org/3.5/library/stdtypes.html#sequence-types-list-tuple-range

Funciones comunes

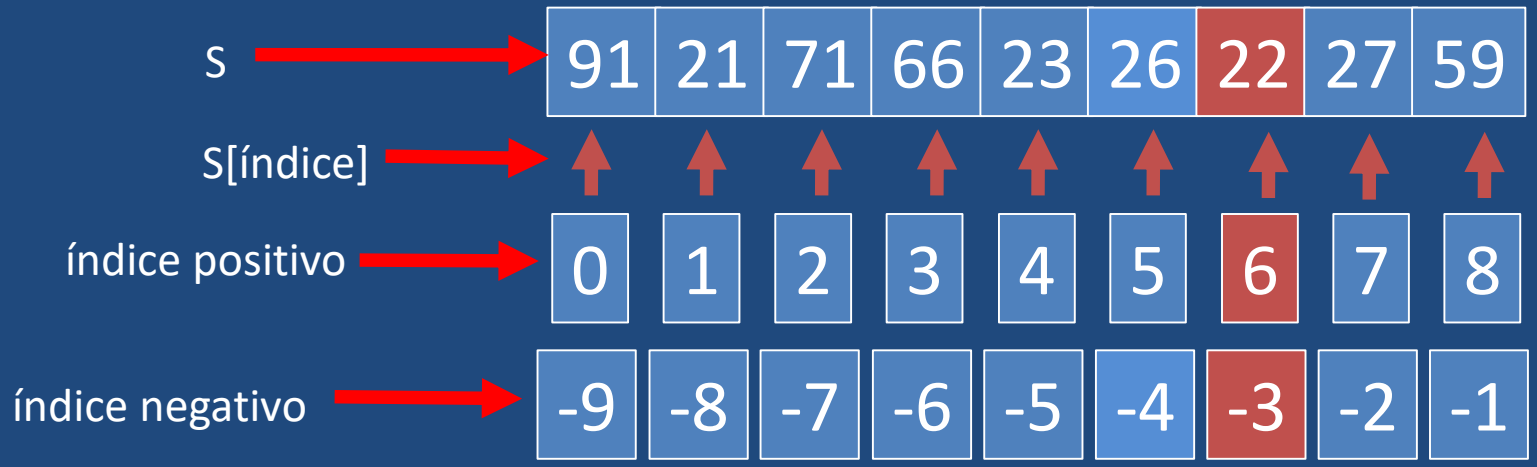
- Ejemplos:

```
>>> a
(31, 50, 43, 34, 46, 25, 28, 24)
>>> 23 in a
False
>>> 51 not in a
True
>>> a + (98,)
(31, 50, 43, 34, 46, 25, 28, 24, 98)
>>> len(a)
8
>>> min(a),max(a)
(24, 50)
>>> a.index(46)
4
>>> a.count(25)
1
```

#1 : https://docs.python.org/3.5/library/stdtypes.html#sequence-types-list-tuple-range

Índices de secuencias

- Los índices de la secuencia numeran cada uno de los elementos de izquierda a derecha y empezando por cero(0). Así se permite acceder a los elementos por su índice y no secuencialmente. Se puede acceder también de derecha a izquierda mediante índices negativos
- Las función len(s) indica el número de elementos de la secuencia, pero no el índice del elemento final (que siempre es len(s)-1)
- Para la secuencia s, s[i] indica el elemento con índice i de la secuencia.
- Así, por ejemplo, s[6] = 22 o s[-3] = 22



Segmentación de secuencias

- Es una operación que se realiza sobre una secuencia o lista. El operador permite extraer partes de la secuencia en una subsecuencia.
- Por ejemplo, de la secuencia 1,2,3,4,5,6,7 extraer sólo los elementos pares 2,4,6
- En la documentación del lenguaje, se explica la sintaxis de los operadores de segmantación (#2)

Operación	Resultado
s[i]	*i-ésimo elemento de s , empezando por* 0
s[i:j]	*Extracción de los elementos de s desde el índice i hasta el j*
s[i:j:k]	*Extracción de los elementos desde el índice i hasta el j en intervalos de k elementos*

(#2) https://docs.python.org/3.5/library/stdtypes.html#mutable-sequence-types

Segmentación de secuencias

- La sintaxis es la siguiente:
- **`s[i]`**
  - i-èsimo elemento de s, empezando por cero (0)
- **`s[i:j]`**
  - extrae la secuencia existente entre los índices i y j
- **`s[i:j:k]`**
  - extrae la secuencia existente entre los índices i y j en incrementos de k

Notas: en los casos de s[i:j] y s[i:j:k]

Si i no se indica, i toma el valor cero → s[:4] es s[0:4]

Si j no se indica, toma le valor de len(s) → s[4:] es s[4:len(s)]

Si k no se indica, toma el valor 1

Ejemplos

- `s[i]` → el elemento de índice i empezando por cero (0)
- Si i es negativo, el índice se recorre desde el final hasta el inicio. Es lo mismo que hacer s[len(s)-i]

```
>>> s
[91, 21, 71, 66, 23, 26, 22, 27, 59, 94]
>>> s[1]
21
>>> s[0]
91
>>> s[6]
22
>>> s[-2]
59
>>> s[-1]
94
```

Ejemplos

- `s[i:j]` → extrae los elementos de la secuencia existente entre los índices i y j-1

  Si i no se indica, i toma el valor cero

  Si j no se indica, toma le valor de len(s)

```
>>> s = [91, 21, 71, 66, 23, 26, 22, 27, 59, 94]
>>> s[1:7]
[21, 71, 66, 23, 26, 22]
>>> s[2:]
[71, 66, 23, 26, 22, 27, 59, 94]
>>> s[:4]
[91, 21, 71, 66]
>>> s[::]
[91, 21, 71, 66, 23, 26, 22, 27, 59, 94]
>>> s[:-2]
[91, 21, 71, 66, 23, 26, 22, 27]
>>> s[-4:]
[22, 27, 59, 94]
>>> s[-8:6]
[71, 66, 23, 26]
>>> s[-8:-2]
[71, 66, 23, 26, 22, 27]
>>>
```

Ejemplos

- **`s[i:j:k]`** → extrae la secuencia existente entre los índices i y j en incrementos de k

Si k no se indica, toma el valor 1

Si k es negativo, los incrementos son negativos. i ha de ser mayor que j

```
>>> s
[91, 21, 71, 66, 23, 26, 22, 27, 59, 94]
>>> s[4:8:2]
[23, 22]
>>> s[8:4:-2]
[59, 22]
>>> s[::-1]
[94, 59, 27, 22, 26, 23, 66, 71, 21, 91]
>>>
```

Ejemplo : girar una secuencia

- La operación `s[::-1]` → devuelve la secuencia en forma especular (el primer elemento es el último y así sucesivamente). Es válido para cualquier tipo de secuencias

```
>>> s
[91, 21, 71, 66, 23, 26, 22, 27, 59, 94]
>>> s[::-1]
[94, 59, 27, 22, 26, 23, 66, 71, 21, 91]
>>> b = ' Eso es una Frase'
>>> b[::-1]
'esarF anu se osE '
>>>
```

Secuencias de datos mutables: Listas

- Listas o secuencias de datos:
  - Se muestran siempre como una secuencia de datos separados por comas y entre corchetes
  - Se puede definir una lista:
    - con la función list()
    - asignando una lista vacía a = '[]' o
    - asignando una lista de valores a = [1,2,'hola']
  - Se pueden añadir elementos y / o modificarlos

```
>>> a = []
>>> a
[]
>>> b = list((2,4,5))
>>> b
[2, 4, 5]
```

```
>>> c = list('hola')
>>> c
['h', 'o', 'l', 'a']
>>> d = [2,7,0.5, 'hola']
>>> d + [2]
[2, 7, 0.5, 'hola', 2]
```

Listas o secuencias de datos mutables

- Listas son secuencias mutables:
  - A diferencias de las tuplas, secuencias de caracteres o de las secuencias de range, las listas pueden ser modificadas en su contenido.

```
>>> lista
[36, 77, 17, 50, 73, 81, 37, 44, 47, 100]
>>> lista[3]=99
>>> lista
[36, 77, 17, 99, 73, 81, 37, 44, 47, 100]
>>> lista.pop(8)
47
>>> lista
[36, 77, 17, 99, 73, 81, 37, 44, 100]
>>> lista += [23,67]
>>> lista
[36, 77, 17, 99, 73, 81, 37, 44, 100, 23, 67]
>>> del lista[6]
>>> lista
[36, 77, 17, 99, 73, 81, 44, 100, 23, 67]
```

Funciones de Listas

- Funciones para listas (#3):
  - A parte de las funciones comunes para todas las secuencias, las listas tienen funciones específicas

Operación	Resultado
s[i] = x	El elemento i de la lista es reemplazado por el valor x
s[i:j] = t	Se substituye la sublista de s entre los índices i y j por las lista t. tanto la sublista como la lista t deben tener la misma longitud.
del s[i:j]	Borra la parte de la lista entre los índices i i j. igual que s[i:j] = []
s[i:j:k] = t	Susutituye los elementos de la sublista indicados en [i:j:k] por la lista t. tanto la sublista como la lista t deben tener la misma longitud.
del s[i:j:k]	Borra los elementos de la sublista indicada en [i:j:k]
s.append(x)	Añade el elemento x a la lista en la última posición (igual que lista += [x])
s.extend(t)	Añade la lista t a la lista s por la derecha (igual que lista += t)
s.clear()	Borra toda la lista s (igual que del s[:])

(#3) https://docs.python.org/3.5/library/stdtypes.html#sequence-types-list-tuple-range

Funciones de Listas

- Funciones para listas (#3):
  - A parte de las funciones comunes para todas las secuencias, las listas tienen funciones específicas

Operación	Resultado
s.copy()	Crea una copia superficial de la lista s (sin alias) (igual que s[:]). La función no admite slicing y se aplica a toda la lista
s *= n	Crea una nueva lista con el contenido de s repetido n veces. Valores ceo o negativos de n producen una lista vacía.
s.insert(i, x)	inserta *x* en *s* en el índice *i* (igual que s[i:i] = [x])
s.pop(i)	Recupera el elemento en la posición i y lo quita también de s. Si el valor de i se omite, i toma el valor -1 y se devuelve y se elimina el último número de la lista
s.remove(x)	Elimina la primera aparición de x en la lista
s.reverse()	invierte los elementos de s en la propia lista

(#3) https://docs.python.org/3.5/library/stdtypes.html#sequence-types-list-tuple-range

Diversos ejemplos:

- Funciones para listas (#3):

```
>>> lista
[36, 77, 17, 99, 73, 81, 44, 100, 23, 67]
>>> lista[9]
67
>>> lista[3:5] = [4,6]
>>> lista
[36, 77, 17, 4, 6, 81, 44, 100, 23, 67]
>>> del lista[3:5]
>>> lista
[36, 77, 17, 81, 44, 100, 23, 67]
```

(#3) https://docs.python.org/3.5/library/stdtypes.html#sequence-types-list-tuple-range

Diversos ejemplos:

- Funciones para listas (#3):

```
>>> lista[2:6:2]
[17, 44]
>>> lista[2:6:2] = [89,54]
>>> lista
[36, 77, 89, 81, 54, 100, 23, 67]
>>> del lista[2:6:2]
>>> lista
[36, 77, 81, 100, 23, 67]
>>> lista.append(65)
>>> lista
[36, 77, 81, 100, 23, 67, 65]
>>> lista.clear()
>>> lista
[]
```

```
>>> a = [1,2,3,4]
>>> t = [5,6,7,8]
>>> a +=t
>>> a
[1, 2, 3, 4, 5, 6, 7, 8]
>>> a
[1, 2, 3, 4, 5, 6, 7, 8]
>>> a = [1,2,3,4]
>>> t = [5,6,7,8]
>>> a. extend(t)
>>> a
[1, 2, 3, 4, 5, 6, 7, 8]
>>>
```

(#3) https://docs.python.org/3.5/library/stdtypes.html#sequence-types-list-tuple-range

Diversos ejemplos:

```
>>> lista = [36, 77, 17, 99, 73, 81, 44, 100, 23, 67]
>>> lista2 = lista.copy()
>>> lista
[36, 77, 17, 99, 73, 81, 44, 100, 23, 67]
>>> lista2
[36, 77, 17, 99, 73, 81, 44, 100, 23, 67]
>>> lista.clear()
>>> lista
[]
>>> lista2
[36, 77, 17, 99, 73, 81, 44, 100, 23, 67]
>>> lista2.extend([34,77,23])
>>> lista2
[36, 77, 17, 99, 73, 81, 44, 100, 23, 67, 34, 77, 23]
```

(#3) https://docs.python.org/3.5/library/stdtypes.html#sequence-types-list-tuple-range

Diversos ejemplos:

```
>>> a = [1,2,3,4]
>>> a
[1, 2, 3, 4]
>>> b = a * 3
>>> b
[1, 2, 3, 4, 1, 2, 3, 4, 1, 2, 3, 4]
>>> b.insert(8,676)
>>> b
[1, 2, 3, 4, 1, 2, 3, 4, 676, 1, 2, 3, 4]
>>> b.pop(6)
3
>>> b.pop()
4
>>> b.remove(676)
>>> b
[1, 2, 3, 4, 1, 2, 4, 1, 2, 3]
>>> b.reverse()
>>> b
[3, 2, 1, 4, 2, 1, 4, 3, 2, 1]
```

(#3) https://docs.python.org/3.5/library/stdtypes.html#sequence-types-list-tuple-range

Funciones de Listas

- Funciones entre listas y strings :
  - Funciones específicas para transformación de strings en listas o transformación de listas en strings

Operación	Resultado
list(s)	Transforma el string en una lista de caracteres
s.split(sep)	Transforma el string en una lista de palabras, usando **sep** como carácter delimitador. Si *sep* se omite, se genera una lista con palabras (el carácter delimitador es un espacio)
s.join(list)	Devuelve una cadena que es la concatenación de las cadenas que hay en la lista. Si un elemento de la lista no es una cadena, se devolverá un error. El separador entre elementos es la cadena que proporciona este método.

Funciones de Listas

- Funciones entre listas y strings :
  - Ejemplos

```
>>> s = 'Una frase de prueba'
>>> fecha = '27/06/1998'
>>> b = list(fecha)
>>> b
['2', '7', '/', '0', '6', '/', '1', '9', '9', '8']
>>> c = s.split()
>>> c
['Una', 'frase', 'de', 'prueba']
>>> data = fecha.split('/')
>>> data
['27', '06', '1998']
>>> passw = ['3','F','k',')','-','Q','6','H']
>>> clave = ''
>>> clave.join(passw)
'3Fk)-Q6H'
```

Aliasing y copia

- Las variables hacen referencia a objetos que se encuentran en memoria.
- Si asignamos una variable a otra, ambas variables se referirán al mismo objeto
- Lo que queremos es que, al hacer una copia de una variable de cualquier tipo, ésta sea totalmente independiente de la original.
- La función id() devuelve el identificador del objeto. Si son iguales, es que ambas variables apuntan al mismo objeto.
- Debido a que la misma lista tiene dos nombres diferentes, a y b, decimos que tiene dos "alias".
- Los cambios realizados en una variable afectan en la otra. (#4)

```
>>> a = [18,23,67]
>>> b = a
>>> id(a)
2356926373384
>>> id(b)
2356926373384
>>> a is b
True
```

```
>>> a [0] = 10
>>> a
[10, 23, 67]
>>> b
[10, 23, 67]
>>>
```

En el ejemplo se puede ver que a y b se refieren a la misma lista después de ejecutar la asignación b = a

(#4) vea el ejemplo en http://www.interactivepython.org/runestone/static/thinkcspy/Lists/Aliasing.html

Aliasing y copia

- Para evitar aliasing en las copias de las listas, deberemos realizar una acción de "copia" específica :

```
>>> a = [18,23,67]
>>> b = a.copy()
>>> b[0] =34
>>> b
[34, 23, 67]
>>> a
[18, 23, 67]
```

```
>>> a = [18,23,67]
>>> b = a[:]
>>> b[0] = 55
>>> b
[55, 23, 67]
>>> a
[18, 23, 67]
```

Ordenación de listas: el método sort()

- Una de las funciones específicas de listas es el método sort() incluido en la variable lista.
  - El método sort() realiza la ordenación sobre la misma lista resultando como resultado la misma lista ordenada
  - El método sort() acepta dos argumentos :

```
list.sort(key=None, reverse=False)
```

Operación	Resultado
key	*Función. key especifica una función de un solo parámetro que es usada para extaer la clave de ordenación para cada elemento de la lista (por ejemplo, key=str.lower para indicar que ordena según las minúsculas de los elementos string de la secuencia que ordena). El valor de None indica que la lista se ordena según los elementos de la secuencia sin calcular claves por separado.*
reverse	*Boolean. Si es True, ordena de mayor a menor. Si es False o se omite, ordena de menor a mayor*

```
>>> a
[5, 9, 0, 9, 8, 0, 4, 7, 6, 7, 0, 1, 7, 5, 6, 6, 2, 5, 9, 6]
>>> a.sort(key = lambda x:(x%2)*100+x)
>>> a
[0, 0, 0, 2, 4, 6, 6, 6, 6, 8, 1, 5, 5, 5, 7, 7, 7, 9, 9, 9]
```

(#5) https://docs.python.org/3.5/library/stdtypes.html#sequence-types-list-tuple-range

Ordenación de listas: la función sorted()

- La función predeterminada sorted(lista) devuelve una copia de la lista ordenada. Dicha función no modifica la ordenación original de la lista

```
>>> a = [5, 9, 0, 9, 8, 0, 4, 7, 6, 7, 0, 1, 7, 5, 6, 6, 2, 5, 9, 6]
>>> sorted(a,key = lambda x:(x%2)*100+x)
[0, 0, 0, 2, 4, 6, 6, 6, 6, 8, 1, 5, 5, 5, 7, 7, 7, 9, 9, 9]
>>> a
[5, 9, 0, 9, 8, 0, 4, 7, 6, 7, 0, 1, 7, 5, 6, 6, 2, 5, 9, 6]
```

(#5) https://docs.python.org/3.5/library/stdtypes.html#sequence-types-list-tuple-range

Ejercicios

- Crear una lista con números aleatorios (Rec)
- Suma de los elementos de una lista (Rec)
- Producto escalar de una lista (Rec)
- Mirar si la lista es creciente (Búsq)
- El primer negativo (Búsq)
- Mayor local (Búsq)

(#5) https://docs.python.org/3.5/library/stdtypes.html#sequence-types-list-tuple-range