SISTEMAS DE GESTIÓN DE BASES DE DATOS AVANZADOS

V2022

Estructuras de datos

Listas

Python tiene varios tipos de datos *compuestos*, usados para agrupar otros valores. El más versátil es la *lista*, la cual puede ser escrita como una lista de valores separados por coma (ítems) entre corchetes. No es necesario que los ítems de una lista tengan todos el mismo tipo

```
>>> a = ['pan', 'huevos', 100, 1234]
>>> a
['pan', 'huevos', 100, 1234]
```

Como los índices de las cadenas de texto, los índices de las listas comienzan en 0, y las listas pueden ser rebanadas, concatenadas y todo lo demás:

```
>>> a[0]
'pan'
>>> a[3]
1234
>>> a[-2]
100
>>> a[1:-1]
['huevos', 100]
>>> a[:2] + ['carne', 2*2]
['pan', 'huevos', 'carne', 4]
>>> 3*a[:3] + ['Boo!']
['pan', 'huevos', 100, 'pan', 'huevos', 100, 'pan', 'huevos', 100,
'Boo!']
```

Estructuras de datos

Listas

Todas las operaciones de rebanado devuelven una nueva lista conteniendo los elementos pedidos. Esto significa que la siguiente rebanada devuelve una copia superficial de la lista *a*:

```
>>> a[:]
['pan', 'huevos', 100, 1234]
```

A diferencia de las cadenas de texto, que son *inmutables*, es posible cambiar un elemento individual de una lista:

```
>>> a
['pan', 'huevos', 100, 1234]
>>> a[2] = a[2] + 23
>>> a
['pan', 'huevos', 123, 1234]
```

Estructuras de datos

Listas

El tipo de dato lista tiene algunos métodos más. Aquí están todos los métodos de los objetos lista:

list.**append**(*x*) **>>** Agrega un ítem al final de la lista; equivale a `a[len(a):] = [x]`.

list.**extend**(*L*) **>>** Extiende la lista agregándole todos los ítems de la lista dada; equivale a `a[len(a):] = L`.

list.**insert**(*i*, *x*) **>>** Inserta un ítem en una posición dada. El primer argumento es el índice del ítem delante del cual se insertará, por lo tanto `a.insert(0, x)` inserta al principio de la lista, y `a.insert(len(a), x)` equivale a `a.append(x)`.

list.**remove**(*x*) **>>** Quita el primer ítem de la lista cuyo valor sea *x*. Es un error si no existe tal ítem.

list.**pop**([*i*]) **>>** Quita el ítem en la posición dada de la lista, y lo devuelve. Si no se especifica un índice, `a.pop()` quita y devuelve el último ítem de la lista. (Los corchetes que encierran a *i* en la firma del método denotan que el parámetro es opcional, no que deberías escribir corchetes en esa posición.

list.**index**(*x*) **>>** Devuelve el índice en la lista del primer ítem cuyo valor sea *x*. Es un error si no existe tal ítem.

list.**count**(*x*) **>>** Devuelve el número de veces que *x* aparece en la lista.

list.**sort**() **>>** Ordena los ítems de la lista, in situ.

list.**reverse**() **>>** Invierte los elementos de la lista, in situ.

Un ejemplo que usa la mayoría de los métodos de lista:

```
>>> a = [66.25, 333, 333, 1, 1234.5]
>>> print a.count(333), a.count(66.25), a.count('x')
2 1 0
>>> a.insert(2, -1)
>>> a.append(333)
>>> a
[66.25, 333, -1, 333, 1, 1234.5, 333]
>>> a.index(333)
1
>>> a.remove(333)
>>> a
[66.25, -1, 333, 1, 1234.5, 333]
>>> a.reverse()
>>> a
[333, 1234.5, 1, 333, -1, 66.25]
>>> a.sort()
>>> a
[-1, 1, 66.25, 333, 333, 1234.5]
```

Estructuras de datos

Listas

```
>>> a = [66.25, 333, 333, 1, 1234.5]
>>> print a.count(333), a.count(66.25), a.count('x')
2 1 0
>>> a.insert(2, -1)
>>> a.append(333)
>>> a
[66.25, 333, -1, 333, 1, 1234.5, 333]
>>> a.index(333)
1
>>> a.remove(333)
>>> a
[66.25, -1, 333, 1, 1234.5, 333]
>>> a.reverse()
>>> a
[333, 1234.5, 1, 333, -1, 66.25]
>>> a.sort()
>>> a
[-1, 1, 66.25, 333, 333, 1234.5]
```

Estructuras de datos

Listas

Listas como Pilas

Los métodos de lista hacen que resulte muy fácil usar una lista como una pila, donde el último elemento añadido es el primer elemento retirado (“último en entrar, primero en salir”). Para agregar un ítem a la cima de la pila, use append(). Para retirar un ítem de la cima de la pila, use pop() sin un índice explícito. Por ejemplo:

```
>>> stack = [3, 4, 5]
>>> stack.append(6)
>>> stack.append(7)
>>> stack
[3, 4, 5, 6, 7]
>>> stack.pop()
7
>>> stack
[3, 4, 5, 6]
>>> stack.pop()
6
>>> stack.pop()
5
>>> stack
```

Listas como Colas

También es muy práctico usar una lista como cola, donde el primer elemento que se añade a la cola es el primero en salir (``first-in, first-out'', ``primero en llegar, primero en salir'').

```
>>> cola = ["Eric", "John", "Michael"]
>>> cola.append("Terry") # llega Terry
>>> cola.append("Graham") # llega Graham
>>> cola.pop(0)
'Eric'
>>> cola.pop(0)
'John'
>>> cola
['Michael', 'Terry', 'Graham']
```

Estructuras de datos

Listas

Conjuntos

Python también incluye un tipo de dato para conjuntos. Un conjunto es una colección no ordenada y sin elementos repetidos. Los usos básicos de éstos incluyen verificación de pertenencia y eliminación de entradas duplicadas. Los conjuntos también soportan operaciones matemáticas como la unión, intersección, diferencia, y diferencia simétrica.

```
>>> canasta = ['manzana', 'naranja', 'manzana', 'pera', 'naranja',
'banana']
>>> fruta = set(canasta)               # crea un conjunto sin
repetidos
>>> fruta
set(['pera', 'manzana', 'banana', 'naranja'])
>>> 'naranja' in fruta                 # verificación de
pertenencia rápida
True
>>> 'yerba' in fruta
False

>>> # veamos las operaciones para las letras únicas de dos palabras
...
>>> a = set('abracadabra')
>>> b = set('alacazam')
>>> a                                  # letras únicas en a
set(['a', 'r', 'b', 'c', 'd'])
>>> a - b                              # letras en a pero no en b
set(['r', 'b', 'd'])
>>> a | b                              # letras en a o en b
set(['a', 'c', 'b', 'd', 'm', 'l', 'r', 'z'])
>>> a & b                              # letras en a y en b
set(['a', 'c'])
>>> a ^ b                              # letras en a o b pero no en
ambos
set(['b', 'd', 'm', 'l', 'r', 'z'])
```

Estructuras de datos

Tuplas

Es un tipo de secuencia estándar, consiste de un número de valores separados por comas.
ver, en la salida las tuplas siempre se encierran entre paréntesis, para que las tuplas anidadas puedan interpretarse correctamente; pueden ingresarse con o sin paréntesis, aunque a menudo los paréntesis son necesarios de todas formas (si la tupla es parte de una expresión más grande).
Las tuplas, al igual que las cadenas, son inmutables: no es posible asignar a los ítems individuales de una tupla

```
>>> t = 12345, 54321, 'hola!'
>>> t[0]
12345
>>> t
(12345, 54321, 'hola!')
>>> # Las tuplas pueden anidarse:
... u = t, (1, 2, 3, 4, 5)
>>> u
((12345, 54321, 'hola!'), (1, 2, 3, 4, 5))
```

```
>>> vacia = ()
>>> singleton = 'hola',    # <-- notar la coma al final
>>> len(vacia)
0
>>> len(singleton)
1
>>> singleton
('hola',)
```

Estructuras de datos

Diccionarios

Los diccionarios se encuentran a veces en otros lenguajes como “memorias asociativas” o “arreglos asociativos”. A diferencia de las secuencias, que se indexan mediante un rango numérico, los diccionarios se indexan con claves, que pueden ser cualquier tipo inmutable; las cadenas y números siempre pueden ser claves

Lo mejor es pensar en un diccionario como un conjunto no ordenado de pares clave: valor, con el requerimiento de que las claves sean únicas (dentro de un diccionario en particular).

Las operaciones principales sobre un diccionario son guardar un valor con una clave y extraer ese valor dada la clave. También es posible borrar un par clave:valor con del. Si usás una clave que ya está en uso para guardar un valor, el valor que estaba asociado con esa clave se pierde. Es un error extraer un valor usando una clave no existente

El método keys() de un diccionario devuelve una lista de todas las claves en uso de ese diccionario

```
>>> tel = {'jack': 4098, 'sape': 4139}
>>> tel['guido'] = 4127
>>> tel
{'sape': 4139, 'jack': 4098, 'guido': 4127}
>>> tel['jack']
4098
>>> del tel['sape']
>>> tel['irv'] = 4127
>>> tel
{'jack': 4098, 'irv': 4127, 'guido': 4127}
>>> tel.keys()
['jack', 'irv', 'guido']
>>> 'guido' in tel
True
```

Estructuras de datos

Diccionarios

Iteraciones

Cuando iteramos sobre diccionarios, se pueden obtener al mismo tiempo la clave y su valor correspondiente usando el método iteritems().

```
>>> caballeros = {'gallahad': 'el puro', 'robin': 'el valiente'}
>>> for k, v in caballeros.iteritems():
...     print k, v
...
gallahad el puro
robin el valiente
```

Ejercicios

Genere un programa que :

A- imprima un menú de opciones desde una lista:

Opciones: 1-Multiplicacion de 2 números, 2- Suma y multiplicación de 3 números, 3- Numero positivo o negativo, 4-Par o impar, 99 - Fin

B- Solicite el ingreso de una opción

C- Dada la opción ejecutar la porción de programa que corresponda:

1. Hacer un programa que permita ingresar dos números, y muestre la multiplicación de ambos
2. Hacer un programa que permita ingresar tres números, y muestre en pantalla tanto la suma como la multiplicación de ellos
3. Confeccione un programa que lea un número e indique si este es positivo o negativo
4. Confeccione un programa que lea un número e indique si este es par o impar